NPS55-85-028

NAVAL POSTGRADUATE SCHOOL

Monterey, California

TECHNICAL

PROGRAM FOR THE SIMULTANEOUS ESTIMATION OF DISPLACEMENT AND ORIENTATION CORRECTIONS FOR SEVERAL SHORT BASE LINE ARRAYS

ROBERT R. READ

NOVEMBER 1985

Approved for public release; distribution unlimited.

Prepared for:
Naval Undersea Warfare Engineering Station
Keyport, WA 98345

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. H. Shumaker
Superintendent

D. A. Schrady
Provost

This report was prepared by:

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NPS55-85-028	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Naval Undersea Warfare Engineering Station
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5100	7b. ADDRESS (*City, State, and ZIP Code*) Keyport, WA 98345
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (*Include Security Classification*)
PROGRAM FOR THE SIMULTANEOUS ESTIMATION OF DISPLACEMENT AND ORIENTATION CORRECTIONS FOR SEVERAL SHORT BASE LINE ARRAYS

12. PERSONAL AUTHOR(S)
Read, Robert R.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	1985 November	70

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Least Squares, Calibration, Multivariate Estimation, Multivariate Regression

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

This report presents the mathematical support and algorithms for the least square estimation of multiple array displacement and orientation corrections. Two situations are treated: (i) cross over data collected in the array overlap region; and (ii) matching of first principal components straight lines for use when there is no replication of data in the array overlap region. Fortran source codes are provided for the first situation.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Robert R. Read	22b. TELEPHONE (*Include Area Code*) (408)646-2382 22c. OFFICE SYMBOL Code 55Re

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

I. INTRODUCTION

The methodology presented here is concerned with the calibration (more precisely, the maintenance of calibration) of a three-dimensional tracking system. The individual sensor arrays are the short base line arrays that produce 3-D track for NUWES. Our task is to consider the coherence of track produced by two arrays in the regions of array overlap. These regions make the continuous tracking of a target achievable. Thus many arrays may contribute to the production of track for a single target and the integration of the various track segments provided by the individual arrays is the main problem in the maintenance of calibration.

An individual track segment produced by a single array is described originally in the local coordinate system of that array. Such a segment must be transformed to the Range coordinate system and integrated with other transformed segments to form a path for the target. These transformations are assumed to be linear and require two kinds of input; (i) the location of each array in the Range coordinate system; and (ii) the orientation of the local coordinate system relative to the Range orientation. In underwater tracking the location and orientation of a local coordinate system must be determined remotely. It is inferred from synchronously timed track segments.

Each array has four sonar receivers located at four of the corners of a rigid cube. The target is a torpedo (or other

self-propelled vehicle). It has a "pinger" attached to it which emits a sound wave at regularly timed intervals called point counts. The position of the target at a given point count is determined from the sound wave's time of arrival differential at the four corners containing the receivers. These signals are transmitted over cables to a central computer.

The functioning of the Range's tracking system requires knowledg of the location and orientation of each array. Moreover, once these values are established, they must be monitored regularly to guard against slippages of various forms, i.e., calibration must be maintained.

There are a number of instrumentation, measurement and environmental problems associated with this type of system, but they are of no concern in the present work. Our goal is restricted to estimating array positions and orientations. Mathematically the former is a three-dimensional vector and the latter is a three by three coefficient matrix that is constrained to be orthonormal, that is, a rigid rotation of an array's cube in three space.

This model for describing the changes in processing the local track from an array would be exact if the speed of sound in water we constant. Such is not quite the case. This speed increases with depth and is assumed to be homogeneous in each horizontal layer of water. Because of the depth gradient there is a non-linear correction term in the vertical position of the object. Generally it is not very large. If an array had sunk fifty feet deeper in a muddy bottom and a typical speed with depth gradient were used then sound ray tracing computations show that the change in

position of a target some 3000 feet away is within a foot of that computed using the linear model. For NUWES purposes it is safe to use the linear model to detect slippage. If the non-linear term were suspected of being important then the estimation methodology could be iterated. That is, the local coordinate system could be corrected using the linear methodology followed by a recomputation of the target's track. Then a reexamination of the consequences of this could be used to decide whether the local coordinate system should be corrected again.

It will be seen that the corrections estimated for an array are not absolute but relative to the other arrays in the system. Because of this it is convenient to assume that the location and orientation of at least one array, relative to the Range, were already established. This fact has implications in the calibration maintenance problem. That is, the user places faith in the current official positioning of one or more arrays. Then possible changes in the positioning of others are estimated. If the user chooses to change his selection of this "anchor" array, the estimated changes for the others must be adjusted mathematically.

The regions of overlap for two or more arrays will be called crossover regions. Crossover data refers to the tracks (in Range coordinates) in the crossover region which are produced by two or more arrays for a common set of point counts (i.e., time points). The estimation of array positioning changes is performed by attempting to make crossover data agree as much as possible in some global sense. We adopt the principle of least squares to serve in this role.

Experience with the least squares approach has revealed some occasional shortcomings. There exist cases for which the least square objective function is a rather flat surface as a function of the parameters that describe the changes. (This has occurred only when the number of arrays in a problem is two.) Since this can result in rather large estimated changes a second method, called the principal components method, has been developed. It involves the fitting of straight lines to the track segments in the crossover region, and this chosen array positions changes that will bring these lines together. It has worked well for the cases that cause trouble in the least squares approach.

A brief comparison of the two methods follows: The principal components method does not require crossover data, i.e., two or more versions of track for a common set of point counts. It does require that the two segments of track being matched should be straight. Also, if the data are not literally of the crossover type, then one must also estimate the distance between the segments being aligned. The least squares approach requires crossover data, but allows the target to be maneuvering in the crossover regio

The paper is organized as follows: The second section contains the development of the least squares method for the simplified case of two arrays producing track in a single crossover region. The main features of the method can be presented in this setting. In the third section the application of these ideas is extended to the general case of multiple arrays and a multiple set of crossover regions. After that we develop the principal component method--again treating the two array problem first.

A Fortran 77 program, KEYMAIN, has been developed to produce the correction estimates from crossover data. The source codes for this program and all subroutines are contained in the Appendix.

II. MATHEMATICAL PRELIMINARIES

In the formulas that follow z will always be a scalar, A and V column vectors, and B a square matrix. The quantities $\partial z/\partial C$ will have the same algebraic structure as C and the elements will be the partial derivatives of z with respect to the elements of C. The superscript T refers to matrix transpose. The following formulas are recorded:

$$\text{If } z = V^T A \text{ then } \partial z/\partial A = V \tag{2.1}$$

$$\text{If } z = A^T A \text{ then } \partial z/\partial A = 2A \tag{2.2}$$

$$\text{If } z = V^T B A \text{ then } \partial z/\partial B = VA^T \tag{2.3}$$

$$\text{If } z = V^T B^T V \text{ then } \partial z/\partial B = 2BVV^T \tag{2.4}$$

Also the trace of a matrix (sum of the diagonal elements) will play a major role. If B and C are square matrices of the same order, liberal use will be made of the facts that

$$\text{Trace}(BC) = \text{Trace}(CB) \tag{2.5}$$

and that the trace is a linear operator. The formulas (2.1) to (2.5) can be found in Reference 3, p. 7ff, and they are not difficult to develop directly.

Consider a set of point counts S in a crossover region, and let the crossover data (in Range coordinates) be

$$\underset{\sim}{y}(t) = \begin{Bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \end{Bmatrix} \quad \text{and} \quad \underset{\sim}{x}(t) = \begin{Bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{Bmatrix} \quad \text{for } t \in S$$

provided by two different sensing arrays. Let us agree that the $\underset{\sim}{y}(t)$ data comes from the array whose location and orientation are established and our goal is to check the calibration of the other array. In particular, does there exist a 3 vector $A \neq 0$, and a 3×3 matrix B ($\neq I$, the identity matrix) such that the adjusted track values,

$$\hat{\underset{\sim}{x}}(t) = A + B\,\underset{\sim}{x}(t)\,, \tag{2.6}$$

are in better agreement with the $\underset{\sim}{y}(t)$ than are the unmodified $\underset{\sim}{x}(t)$.

The vector A is related to a displacement of the sensing array and the matrix B is related to a correction of its orientation. If we let $\xi(t)$ be $\underset{\sim}{x}(t)$ in the local coordinate system, α be the location (in Range coordinates) of the array, and β the (orthonormal) orientation adjustment to the local coordinates then

$$\underset{\sim}{x}(t) = \alpha + \beta\xi(t)$$

and

$$\hat{\underset{\sim}{x}}(t) = A + B\alpha + B\beta\xi(t) \tag{2.7}$$

The corrected location and orientation adjustments are $A+B\alpha$ and $B\beta$, respectively.

Our immediate goal is to estimate A and B using the principle of least squares. We will minimize the average square deviation between $\underset{\sim}{y}(t)$ and $A + B\underset{\sim}{x}(t)$ for each point count. Using the squared norm notation, $\|V\|^2 = V^T V$,

$$Q = \underset{t}{\text{Ave}} \; \|\underset{\sim}{y}(t) - A - B\underset{\sim}{x}(t)\|^2 \tag{2.8}$$

and, since what follows involves a modification of the standard multivariate regression development, let us review an outline of the details.

First, the estimator for A is

$$\hat{A} = \bar{\underset{\sim}{y}} - \hat{B}\,\bar{\underset{\sim}{x}} \tag{2.9}$$

where $\bar{\underset{\sim}{y}} = \underset{t}{\text{Ave}}\; \underset{\sim}{y}(t)$ and $\bar{\underset{\sim}{x}} = \underset{t}{\text{Ave}}\; \underset{\sim}{x}(t)$. The proof follows from an expansion of (2.8), namely

$$Q = \underset{t}{\text{Ave}}[\underset{\sim}{y}(t) - B\underset{\sim}{x}(t)]^T[\underset{\sim}{y}(t) - B\underset{\sim}{x}(t)]$$

$$- 2\,\underset{t}{\text{Ave}}[\underset{\sim}{y}(t) - B\underset{\sim}{x}(t)]^T A + A^T A$$

to which we apply the rules (2.1) and (2.2),

$$\frac{\partial Q}{\partial A} = -2\,\underset{t}{\text{Ave}}[\underset{\sim}{y}(t) - B\underset{\sim}{x}(t) - A] \; .$$

Set this equal to zero, solve for A and the result follows.

Next, let us make a change and work with the deviations

$$Y(t) = \underset{\sim}{y}(t) - \bar{\underset{\sim}{y}} \quad \text{and} \quad X(t) = \underset{\sim}{x}(t) - \bar{\underset{\sim}{x}} \tag{2.10}$$

and use (2.9) in (2.8). The result is a simplified appearance for Q,

$$Q = \underset{t}{\text{Ave}} \; \| Y(t) - BX(t) \|^2 \tag{2.11}$$

Also introduce the covariance matrices

$$D_{yY} = \underset{t}{\text{Ave}} \{ Y(t) Y^T(t) \} \qquad D_{yx} = \underset{t}{\text{Ave}} \{ Y(t) X^T(t) \}$$

$$D_{xx} = \underset{t}{\text{Ave}} \{ X(t) X^T(t) \} \tag{2.12}$$

Now the customary estimator for B can be derived readily. Thus

$$Q = \underset{t}{\text{Ave}} \{ Y^T(t) Y(t) - 2Y^T(t) BX(t) + X^T(t) B^T BX(t) \} \tag{2.13}$$

and use (2.3) and (2.4) to get

$$\frac{\partial Q}{\partial B} = -2 \underset{t}{\text{Ave}} \{ Y(t) X^T(t) - BX(t) X^T(t) \}$$

$$= -2 \{ D_{yx} - B D_{xx} \} \tag{2.14}$$

using (2.12). Set this equal to zero and solve for $\hat{B} = D_{yx} D_{xx}^{-1}$ which is the usual multivariate regression estimator, Ref. 1, p. 181.

Generally, the above usual estimator is not the one we should use. Recall that the sensing arrays are rigid cubes. If they have slipped (i.e., moved physically such as by the action of a ship's anchor hooking a cable) then they undergo a displacement and a re-orientation. The matrix B should be orthonormal as is the matrix β in the original calibration. Thus

$$B^T B = I \tag{2.15}$$

and the estimation of B involves the minimization of Q subject to the constraint. The usual estimate would be useful only if there were physical damage to an array, resulting in its behavior as a skewed rather than Cartesion system.

Use of (2.15) and (2.12) in (2.13) allows the rewrite

$$Q = \text{Trace}\{D_{yy} - 2D_{yx}B^T + D_{xx}\} \tag{2.16}$$

and the minimization of Q is tantamount to the maximization of

$$W = \text{Trace}\{D_{yx} B^T\} \quad \text{subject to (2.15)} \tag{2.17}$$

since the trace is a linear operator and B is involved only in the second term of (2.16). The solution of the optimization problem (2.17) and its generalization in Section 4 is the bulk of our effort.

Earlier was stated the need for placing faith in the location and orientation of at least one array, because otherwise the

solution could not be unique. Let us show now, using analysis, why this is the case in the two array, single crossover region problem. Suppose we tried to adjust both data sets, i.e.,

$$\begin{aligned} \hat{\underset{\sim}{y}}(t) &= A_1 + B_1\underset{\sim}{y}(t) \quad \text{and} \\ \hat{\underset{\sim}{x}}(t) &= A_2 + B_2\underset{\sim}{x}(t) \end{aligned} \tag{2.18}$$

when both B_1 and B_2 are constrained to be orthonormal. The least squares objective function has the appearance

$$Q = \underset{t}{\text{Ave}} \left\| A_1 + B_1\underset{\sim}{y}(t) - A_2 - B_2\underset{\sim}{x}(t) \right\|^2 \tag{2.19}$$

It is clear from the development of equation (2.9) that the displacement portion of the optimization can be estimated only relatively. That is, only $A = A_2 - A_1$ can be found uniquely. From the earlier development we can infer that $\hat{A} = \hat{B}_1\bar{\underset{\sim}{y}} - \hat{B}_2\bar{\underset{\sim}{x}}$ and re-express the objective as

$$Q(B_1, B_2) = \underset{t}{\text{Ave}} \left\| B_1 Y(t) - B_2 X(t) \right\|^2 \tag{2.20}$$

But this is the squared lengths of a set of vectors which have been averaged. Since the squared length of any vector is invariant under any orthonormal transformation, say C, it follows that

$$Q(B_1, B_2) = Q(CB_1, CB_2) \ .$$

In particular we can take $C = B_1^T$ (i.e., the inverse of B_1) and then $Q(B_1, B_2) = Q(I, B_1^T)$. The product of orthonormal matrices

is itself orthonormal. We can set $B = B_1^T B_2$ and note that the minimization of Q(I,B) is that of (2.11), the problem we are treating. The geometrical interpretation is that the orientation of a sensing array can be matched only relatively to the other array. It is convenient to assume that the latter orientation is known.

III. SOLUTION ALGORITHM

The matrix B has nine elements, but because of the constraint (2.15) there are only three degrees of freedom. That is, there are three length constraints and three orthogonality constraints. The three remaining degrees of freedom can be expressed in terms of the Euler angles, Ref. 2, p. 250 ff.

Any orthnormal transformation in 3-space may be viewed as the application of three successive rotations, i.e., hold the x_1-axis fixed and execute a rotation in the x_2-x_3 plane; then hold the x_2-axis fixed in its new position and rotate in the x_1-x_3 plane; and finally rotate in the x_1-x_2 plane with the x_3-axis held fixed in its new position. Denote the angles of these three rotations as ϕ_1, ϕ_2, ϕ_3. In navigation work they are called roll, pitch and yaw. These angles are unique only when the order of application of the rotations is specified.

To shorten the writing, let

$$c_i = \cos(\phi_i) \quad \text{and} \quad s_i = \sin(\phi_i) \tag{3.1}$$

for i = 1,2,3, and the individual planar rotations can be expressed as

$$\rho_1 = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & c_1 & -s_1 \\ 0 & s_i & c_i \end{Bmatrix} \quad \rho_2 = \begin{Bmatrix} c_2 & 0 & -s_2 \\ 0 & 1 & 0 \\ s_2 & 0 & c_2 \end{Bmatrix} \tag{3.2}$$

$$\rho_3 = \begin{Bmatrix} c_3 & -s_3 & 0 \\ s_3 & c_3 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$$

and, using the order of application described, let

$$B = \rho_3 \rho_2 \rho_1 \tag{3.3}$$

We make liberal use of the fact that the product of orthonormal matrices is itself orthonormal.

We note in passing that the matrices B_i above are the transpose (and hence inverses) of those that are usually used in studying the motion of aeroplanes, Ref. [2]. Also the order of application is the reverse of the customary one thus making B in (3.3) the inverse of the usual matrix. This choice is appropriate for our work since the angular correction will rotate the local axes back to where they are expected to be.

This done our optimization problem may be stated as

$$\underset{\phi_1,\phi_2,\phi_3}{\text{Maximize}} \ \text{Trace}\{D_{yx} B^T\} \tag{3.4}$$

The solution can be found by treating the problem as three successive two-dimensional problems. To do this let us first characterize how this optimization problem would be solved if the tracking took place in two dimensions.

In the plane, an orthonormal matrix B has one degree of freedom and it is characterized by a rotation through an angle ϕ.

As before, let $c = \cos(\phi)$, $s = \sin(\phi)$ so that

$$B = \begin{Bmatrix} c & -s \\ s & c \end{Bmatrix}$$

and

$$W = \text{Trace}(DB^T) = (D_{11} + D_{22})c + (D_{21} - D_{12}(s) \qquad (3.5)$$

Thus W is itself a sine wave. It has one max and one min, and these are separated by π radians. Since

$$W' = -(D_{11} + D_{22})s + (D_{21} - D_{12})c \quad \text{and}$$

$$W'' = -(D_{11} + D_{22})c - (D_{21} - D_{12})s$$

it is seen easily that the max occurs when

$$s = K(D_{21} - D_{12}), \quad c = K(D_{11} + D_{22}), \quad \text{and} \qquad (3.6)$$

$$K = \{(D_{21} - D_{12})^2 + (D_{11} + D_{22})^2\}^{-1/2}$$

This would be the solution if the tracking problem were a two-dimensional one.

Now we are positioned to treat our three-dimensional tracking problem. Introduce matrices

$$E = D_{yx}\rho_1^T\rho_2^T \quad F = \rho_3^T D_{yx}\rho_1^T \quad G = \rho_2^T\rho_3^T D_{yx} \qquad (3.7)$$

and note that, using (2.5), (2.17) and (3.3),

$$W = \text{Trace}\{D_{yx}B^T\} = \text{Trace}\{E\rho_3^T\}$$

$$= \text{Trace}\{F\rho_2^T\} = \text{Trace}\{G\rho_1^T\} \tag{3.8}$$

These latter three representatives of W allow us to use our knowledge of the solution to the two-dimensional problem. When these three traces are written out in expanded form and compared with (3.5) and (3.6), it is seen that the optimal solution for ϕ_1, ϕ_2, ϕ_3 must have the form

$$s_3 = K_3(E_{21} - E_{12}) \quad \text{and} \quad c_3 = K_3(E_{11} + E_{22})$$

$$s_2 = K_2(F_{31} - F_{13}) \quad \text{and} \quad c_2 = K_2(F_{11} + F_{33}) \tag{3.9}$$

$$s_1 = K_1(G_{32} - G_{23}) \quad \text{and} \quad c_1 = K_1(G_{22} + G_{33})$$

where

$$K_3 = \{(E_{21} - E_{12})^2 + (E_{11} + E_{22})^2\}^{-1/2}$$

$$K_2 = \{(F_{31} - F_{13})^2 + (F_{11} + F_{33})^2\}^{-1/2} \tag{3.10}$$

$$K_1 = \{(G_{32} - G_{23})^2 + (G_{22} + G_{33})^2\}^{-1/2}$$

The above does not provide explicit solutions because $E = E(\phi_1,\phi_2)$, $F = F(\phi_1,\phi_3)$ and $G = G(\phi_2,\phi_3)$. It is necessary to get all three angles satisfying (3.9) and (3.10) at the same time. This means in addition that

$$\nabla W = 0 \tag{3.11}$$

as well. We know that each of the equations

$$\begin{aligned}
\frac{\partial W}{\partial \phi_3} &= \text{Trace } E \frac{\partial \rho_3^T}{\partial \phi_3} = 0 \\
\frac{\partial W}{\partial \phi_2} &= \text{Trace } F \frac{\partial \rho_2^T}{\partial \phi_2} = 0 \\
\frac{\partial W}{\partial \phi_1} &= \text{Trace } G \frac{\partial \rho_1^T}{\partial \phi_1} = 0
\end{aligned} \tag{3.12}$$

has two solutions, one max and one min. It follows that (3.11) has eight (2^3) solutions, one for each of the permutations of the component solutions, and this means one max, one min, and six saddle points. We require an algorithm that converges to the max.

This can be achieved with an iterative process. Take an arbitrary beginning set of values for ϕ_1, ϕ_2, ϕ_3. All $\phi_i = 0$ will do. Compute E for this choice and then compute a new ϕ_3 from the first member of (3.9). Use this value and the original ϕ_1 to compute F and from this compute a new ϕ_2 from the second member of (3.9). Next use these new values for ϕ_2, ϕ_3 and compute G, followed by a new ϕ_3 from the third member of (3.9).

This completes one cycle. At each step the appropriate member of (3.12) was satisfied and with a max. The value of W increased ea time. Moreover, a second cycle beginning with the values ϕ_1, ϕ_2, and ϕ_3 from the first cycle would produce a further increase in W. We know there exists a unique maximum for W and we have an iterative method that increases W at each step. It follows that, by repeating the process, we can come arbitrarily close to this maximum and that will be signaled when all three members of (3.12) are negligibly small in magnitude.

In order to develop intuition that may be useful in understanding the more complicated optimization process addressed in Section 4, let us take pause now and make a heuristic interpretation of our iterative process. The objective function W is a sine wave as a function of each angle ϕ_i (with the other two angles held fixed). As such it is concave in half of this restricte angular domain. It follows that in the full three-dimensional space of (ϕ_1,ϕ_2,ϕ_3) the function W is concave in one eighth of its domain. A more general (and perhaps more rapidly converging) gradient search algorithm should begin in this part of the domain so that when convergence takes place, it is toward a maximum. If the algorithm presented is used, one does not have to worry about this point.

We note in passing that the above technique is not limited to three dimensions. Suppose that our tracking problem was in p dimensions and that B was a $p \times p$ orthonormal matrix. This constraint reduces the degrees of freedom, p^2 in B by p for the

length constraints, and by an additional $p(p-1)/2$ for the orthogonality constraints. Since

$$p^2 - p - p(p-1)/2 = p(p-1)/2$$

we see that $p(p-1)/2$ is the number of angles (or equivalently, product matrices in the analogue of (3.2)) of rotation in B. I.e., it is the number of ways we can choose p-2 axes (from p axes) to be held fixed while a rotation takes place in the planes remaining. The solution structure will have one max, one min, and 2^p-2 saddle points. Our iteration technique will converge to the max.

IV. GENERAL CASE

To treat the general case we must consider several arrays, say K in number, and several crossover regions. Also we must allow for the target to be tracked in a given crossover region either not at all, exactly once, or more than once since it may maneuver back into a given region during a later point count set. Finally the target, during a given point count set in a given crossover region, may be tracked by more than one sensing array. We proceed to develo a notational structure that can handle this fully general case.

Let $S_1, S_2, \ldots, S_R$ represent a collection of R point count sets. It is convenient to assume that to each individual S_s (for $s = 1, \ldots, R$) there is associated a pair or more of sensing arrays. Thus, if only one array tracks the target in a particular crossover region there will be no corresponding point count set in the collection. If exactly two arrays track the target then the particular S_s is well defined. If three or more arrays track the target at about the same time, then things become a little fuzzy because the point count set for one pair of arrays may not be exactly congruent with the point count set of the crossover data of one of the two arrays with a third array, etc. This possible lack of congruence does not affect the computation of the cross covariance, (2.12) as they appear only in pairs. So there is no harm in allowing the sets $S_1, \ldots, S_R$ to duplicate some segments of track. When such duplicatio occurs however, the array pairs must be distinct.

Next, for $i = 1, \ldots, K$, let C_i be the subcollection of $S_1, \ldots, S_R$ which has tracking data from the $i^{\underline{th}}$ array. In this

way we can identify a 3-dimension vector of track data, $\underset{\sim}{X}_i(t,s)$, as being produced by the $i^{\underline{th}}$ array at point count t which belongs to S_s. These quantities exist only if $S_s \in C_i$.

Our goal is to estimate the displacement and re-orientation parameter pairs for each of the K arrays. Earlier, with K = 2, we learned that there is an identifiability problem and it was convenient to assume that the location and orientation of one of the arrays was fixed. The same is true in the general case (i.e., only one array is fixed) provided that the data satisfy a connectivity condition. This condition can be described in the parlance of graph theory. The K sensing arrays form the nodes. An arc exists between two nodes if crossover data exists between them. We wish to consider only connected graphs. What this means is: There is no partition of the nodes into two non-empty sets such that an arc cannot be found connecting a node of one set to a node of the other set.

If this condition of connectedness is not satisfied by our problem, then the overall data must be decomposed into smaller sets so that it does hold for each subset. Such decompositions are unique and each member of the decomposition is to be treated separately. This done we can fix our analysis on the case that involves a connected graph. Here it will be seen that the estimation of all displacements and re-orientation parameters will be unique once that one set (i.e., a set corresponding to a given array) is specified.

Our notation needs to be expanded. Let $\underset{\sim}{X}_i(t;s)$ be the 3-dimensional track, in range coordinates, produced by the $i^{\underline{th}}$

array at point count t belonging to S_s. Our goal is to estimate the location vectors A_i and re-orientation matrices B_i so that (compare (2.6))

$$\hat{\underset{\sim}{x}}_i(t;s) = A_i + B_i \underset{\sim}{x}_i(t;s) \tag{4.1}$$

are as compatible as possible with the overall track of the target in the range. Also let $\bar{\underset{\sim}{x}}_i(s)$ be the vector of averages of the $\underset{\sim}{x}_i(t;s)$ taken over t in S_s, and

$$X_i(t;s) = \underset{\sim}{x}_i(t;s) - \bar{\underset{\sim}{x}}_i(s) \tag{4.2}$$

be the deviations from the mean. Our new objective function can be expressed as an extension of (2.19)

$$Q = \sum_{i<j} \sum_{s} N_s \text{ Ave } \| B_i X_i(t;s) - B_j X_j(t;s) \|^2 \tag{4.3}$$

where the outer double sum is over $1 \leq i < j \leq k$, the inner sum is over $s \in C_i \cap C_j$, and the average is to be taken over $t \in S_s$. The weights N_s = number of point counts in S_s for the array pair (i,j).

We can view (4.3) as

$$Q = Q(B_1, B_2, \ldots, B_K) \tag{4.4}$$

and, because of the connectedness assumption, (4.3) cannot be decomposed into the sum of two terms

$$Q_1(B_{i_1},\ldots,B_{i_p}) \quad \text{and} \quad Q_2(B_{j_1},\ldots,B_{j_r})$$

for which the subscript sets $i_1,\ldots,i_p$ and $j_1,\ldots,j_r$ form a partition of $1,2,\ldots,K$. Now we can reformat the argument used at the end of Section 2. If C is any orthonormal transformation then $Q(CB_1,CB_2,\ldots,CB_K) = Q(B_1,\ldots,B_K)$. Using $C = B_1^T$, say, then

$$Q(B_1,\ldots,B_K) = Q(I,B_1^TB_2,\ldots,B_1^TB_K) \tag{4.5}$$

and, for convenience, it is assumed that the orientation of the first array is known, i.e., $B_1 = I$ in (4.3).

Define covariance matrices

$$D_{ij}(s) = \text{Ave}\{X_i(t;s)X_j^T(t;s)\} \tag{4.6}$$

where the average is taken over the point counts t in S_s. These quantities exist only for $s \in C_i \cap C_j \neq 0$. Then (4.3) may be expanded to the more useful form

$$Q = \text{Trace} \sum_{i<j} \sum_s N_s\{D_{ii}(s) - 2D_{ij}(s)B_j^TB_1 + D_{jj}(s)\} \tag{4.7}$$

and again, because the unknown matrices $\{B_k\}$ appear only in the second term, we may choose to maximize

$$W = \sum_{i<j} \text{Trace}\{D_{ij}B_j^TB_i\} \tag{4.8}$$

for $1 \leq i < j \leq K$ and

$$D_{ij} = \sum_s N_s D_{ij}(s) \quad \text{or} \quad \text{zero} \tag{4.9}$$

according to whether the summation for $s \in C_i \cap C_j$ has content, or is empty.

A word about the computation of $D_{ij}(s)$ in (4.9). It seems wise to use the pooled within groups covariance in those cases for which S_s is the union of rather diverse point counts. An example should make the point. Suppose the array pair (i,j) tracks the target at point counts {1,...,25} and also {86,...,135}. The pooled within groups covariance would compute the two covariances separately and then combine them into one using a weighted (by sample size) average. In our example, the weights would be 25/75 and 40/75.

For each fixed value of r (r = 1,...,K) the right hand side of (4.8) contains an expression of the form

$$W_r = \sum_{i<r} \text{Trace}\{D_{ir} B_r^T B_i\} + \sum_{j>r} \text{Trace}\{D_{rj} B_j^T B_r\}$$

$$= \text{Trace}\{\sum_{i<r} B_i D_{ir} B_r^T + \sum_{j>r} D_{rj} B_j^T B_r\} \tag{4.10}$$

To shorten the writing let

$$E_r = \sum_{i<r} B_i D_{ir} \quad \text{and} \quad F_r = \sum_{j>r} D_{rj} B_j^T \tag{4.11}$$

and use the fact that $\text{Trace}\{F_r B_r\} = \text{Trace}\{B_r^T F_r^T\} = \text{Trace}\{F_r^T B_r^T\}$. Then

$$W_r = \text{Trace}\{(E_r + F_r^T) B_r^T\} \tag{4.12}$$

Also it is interesting to note that each term of (4.8) appears in two and only two distinct W_r. It follows that

$$\sum_1^k W_r = 2W \tag{4.13}$$

Now the vector of partial derivatives of W with respect to the three Euler angles in B_r, that is ϕ_{r1}, ϕ_{r2}, ϕ_{r3}, is the same as the gradient of W_r (with $B_1,\ldots,B_{r-1},B_{r+1},\ldots,B_k$ held fixed). Moreover, the structure of (4.12) is the same as that of (3.4). We know from Section 2 that ∇W_r has eight zeros, exactly one of which corresponds to a max, and we have an algorithm that converges to that max.

The analysis above leads to the construction of a gradient search that can ferret out the max of W. To fix one array we set $B_1 = I$ and keep this throughout. Choose starting matrices for the set $B_2,\ldots,B_K$. Compute $E_r = E_r(B_1,\ldots,B_{r-1})$ and $F_r = F_r(B_{r+1},\ldots,B_K)$ for each $r = 1,2,\ldots,K$ ($E_1 \equiv 0 \equiv F_K$). Use the algorithm in Section 2 applied to (4.12), for each r, to produce the new Euler angles, i.e., the system $\{B_r\}$. Use these to recompute the $\{E_r,F_r\}$ and repeat. Stop when the gradients of all W_r are sufficiently small in magnitude. Each W_r will

be at a local maximum for its argument B_r with all of the other orientation matrices held fixed.

There are some open questions about the convergence properties of this algorithm. From an empirical point of view it has appeared to work successfully. Convergence is not monotone however. In none of our test problems have the $\{B_i\}$ departed notably from identity matrices, and the use of all $B_i = I$ to initialize the algorithm has provided satisfactory results. For one run the initialization was chosen at random (in the 3(K-1) dimensional space of Euler angles) and the time to convergence was excessive. We know that W has many saddle points and that it is a concave function in only a small part of its domain.

Methods for initializing the algorithm need to be developed. The following idea may hold some promise: Referring to each term of (4.8),

$$\text{Trace}\{D_{ij}B_j^T B_i\} = \text{Trace}\{D_{ij}B_{ij}^{*T}\},$$

where $B_{ij}^* = B_i B_j^T$, the solution algorithm of Section 3 can be used to check whether B_{ij}^* is near to the identity. If so, initialize all $B_i = I$. If not, then study of the resulting $\{B_{ij}^*\}$ may lead to a good selection.

Let us return to the objective function Q of (4.3) and use (4.1) in it for purposes of estimating the location parameters $\{A_r\}$. Formally we have

$$Q = \sum_{i<j} \sum_s N_s \text{ Ave } \| A_i + B_i \underset{\sim}{x}_i(t;s) - A_j - B_j \underset{\sim}{x}_j(t;s) \|^2 \qquad (4.14)$$

where $1 \leq i < j \leq K$; $s \in C_i \cap C_j$; the average is taken over the points counts t in S_s; and N_s = number of point counts in S_s. Again it is useful to use terms

$$Q_r = \sum_{i<r} \sum_{s} N_s \text{ Ave } \| A_i + B_i \underset{\sim}{x}_i(t;s) - A_r - B_r \underset{\sim}{x}_i(t;s) \|^2$$

$$+ \sum_{j<r} \sum_{s} N_s \text{ Ave } \| A_r + B_r \underset{\sim}{x}_r(t;s) - A_j - B_j \underset{\sim}{x}_j(t;s) \|^2 \qquad (4.15)$$

and recognize that $\sum_{r=1}^{K} Q_r = 2Q$. In a manner similar to the one that produced (2.7) we can develop

$$\frac{\partial Q_r}{\partial A_r} = -2 \sum_{i<r} \sum_{s} N_s \{ B_i \bar{\underset{\sim}{x}}_i(s) - B_r \bar{\underset{\sim}{x}}_r(s) - (A_r - A_i) \}$$

$$+ 2 \sum_{j>r} \sum_{s} N_s \{ B_r \bar{\underset{\sim}{x}}_r(s) - B_j \bar{\underset{\sim}{x}}_j(s) - (A_j - A_r) \} \qquad (4.16)$$

Setting (4.16) equal to zero results in a 3K linear system in the $A_1, \ldots, A_K$. Let $M_{ij} = \sum_{s} N_s$ for $s \in C_i \cap C_j$. Then the left hand side of the linear system is

$$- \sum_{i<r} M_{ir} A_i + \{ \sum_{i<r} M_{ir} + \sum_{j>r} M_{rj} \} A_r - \sum_{j>r} M_{rj} A_j \qquad (4.17)$$

and the right hand side is

$$\sum_{i<r} B_i \sum_s N_{s\sim i} \bar{x}_{\cdot i}(s) + \sum_{j>r} B_j \sum_s N_{s\sim j} \bar{x}_{\cdot j}(s)$$

$$- B_r \{ \sum_{i<r} \sum_s N_{s\sim r} \bar{x}_{\cdot r}(s) + \sum_{j>r} \sum_s N_{s\sim r} \bar{x}_{\cdot r}(s) \} \qquad (4.18)$$

Notice that the coefficient matrix in (4.17) is the same for all three components of the $\{A_r\}$. Also the columns of the matrix add to zero so that its rank is K-1. It follows that the system is underdetermined. We anticipated this. Setting $A_1 = 0$ will allow a unique solution for $A_2,\ldots,A_K$. Of course, this choice corresponds to the assumption made earlier that the location and orientation of the first array is known.

V. PRINCIPAL COMPONENTS METHOD

Experience with the least square method (two arrays, Section 2) in the area of underwater tracking has revealed some instabilities. Several cases were identified in which the surface W contained some rather extensive relatively flat regions as a function of the three Euler angles. More specifically, it is possible to move from the maximizing point to a saddle point and lose less than one percent of the value of W. Moreover the effect of using the maximizing point can be to assign an absurd relocation geometry to the position of the sensor (e.g., turned upside down and suspended without support somewhere in, or above, the water). In these cases use of the saddle point involved only minor relocation of the sensor. Clearly this is an unsatisfactory state of affairs.

These anomalies can happen, and there is not yet any fully reliable way to identify the conditions under which they may occur. They seem to be related to the inherent variability of the tracking data coupled with the point by point crossover data matching approach utilized by the least square methodology. It may be that certain naturally occurring geometric conditions (e.g., straight line structure of track) may act as a catalyst for this phenomenon.

An alternative methodology has been developed and it appears to avoid producing the absurd solutions mentioned above. It is based on the matching of the first principal components of the covariance matrices of the tracking data.

Let us take a moment to describe the notion of first principal component. Suppose we want to fit a segment of straight line to a track of data $\{\underset{\sim}{x}(t); t \in S\}$. Further suppose we use the minimum sum of squared projections (orthogonal) as the mathematical criterion for choosing the line. The solution is well known (Ref. 1, p. 272 ff). Let λ be the largest eigenvalue of the covariance matrix D_{xx} and let q be an eigenvector associated with λ. That is, any solution of

$$D_{xx}q = \lambda q \tag{5.1}$$

The following facts are stated without proof:

(i) The vector q is a set of direction numbers for the desired straight line.

(ii) The line passes through the centroid $\underset{\sim}{\bar{x}}$.

(iii) The projected track of $\underset{\sim}{x}(t)$ onto this straight line are the scalar values $u(t) = q^T \underset{\sim}{x}(t)$.

These properties can be exploited for our current problem in the following way. Let p be a first principal component for D_{yy} and q play the same role for D_{xx}. Both are assumed to be scaled to have length one:

$$\sum p_i^2 = 1 = \sum q_i^2$$

Adopt as the optimizing criterion: Choose B orthonormal so that

$$p = Bq \tag{5.2}$$

That is, the re-orientation matrix is chosen so that the two fitted straight line segments have the same direction numbers.

It appears that the mathematical problem, $p = Bq$, is most easily solved using a constructive method. If $p = q$ then $B = I$. Barring this, proceed as follows:

1) Let θ be the angle between p and q.

2) Form an orthonormal basis $H = \{h_1, h_2, h_3\}$ such that $h_1 = q$; h_2 is in the plane of p and q but orthogonal to h_1; h_3 completes the basis.

3) Letting $c = \cos(\theta)$ and $s = \sqrt{1 - c^2}$, we can use the Gram Schmidt process and take

$$h_2 = \frac{(q - cp)}{s}$$

4) Notice that

$$Hq = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad \text{and} \quad Hp = \begin{pmatrix} c \\ s \\ 0 \end{pmatrix}$$

and that Hp can be rotated through an (Eulerian) angle θ into Hq by applying the matrix $\rho_3^T(\theta)$, see (3.2).

5) Since $\rho_3 Hp = Hq$ and both ρ_3 and H are orthonormal matrices, it follows that $p = H^T \rho_3^T Hq$. Thus

$$B = H^T \rho_3^T H \tag{5.3}$$

Notice that this method does not require crossover data in the strict sense, i.e., matched pairs of track from two arrays for a common set of point counts. The matrix D_{yx} is not needed

and D_{yy} and D_{xx} can be computed without this requirement. However the stability of the principal components p and q require that the two pieces of track be rather straight, i.e., no curvilinear bias.

If the input data are not crossover data, there must be an adjustment in the way that we estimate the displacement parameter A. The formula (2.9) will not be valid because the $\bar{\underset{\sim}{x}}$ and $\bar{\underset{\sim}{y}}$ values do not refer to the same time (i.e., point count). Our immediate goal is to estimate the distance δ traveled by the target from its mean position $\bar{\underset{\sim}{y}}$ at some time $\bar{t}_y$ to its new estimated position $A + B\bar{\underset{\sim}{x}}$ at some later time $\bar{t}_x$. Let us assume that the two point count sets (one for the $\{\underset{\sim}{y}\}$ and one for the $\{\underset{\sim}{x}\}$) are mutually exclusive and represent equally spaced time values. Since the target's path is straight we can average the times in the two point count sets to obtain values for $\bar{t}_y$ and $\bar{t}_x$. Then one can form the sets of projections

$$u(t) = q^T \underset{\sim}{x}(t) \qquad \text{and} \qquad v(t) = p^T \underset{\sim}{y}(t) \tag{5.4}$$

and use their successive differences to get an estimate of the target's speed.

The distance δ will be the speed multiplied by the time increment $\bar{t}_x - \bar{t}_y$.

This done, it is claimed that

$$A = \bar{\underset{\sim}{y}} - B\bar{\underset{\sim}{x}} + \delta p \tag{5.5}$$

Proof: The lineal distance δ can be represented as

$$p^T(A + B\bar{x} - \bar{y}) = \delta$$

from which it follows that $p^TA = p^T(\bar{y} - B\bar{x}) + \delta$. We can view p as the first column of a seld adjoint matrix P, and letting $\underset{\sim}{\delta} = (\delta,0,0)^T$ we can express our equation in the form

$$P^TA = P^T(\bar{y} - B\bar{x}) + \underset{\sim}{\delta} \qquad (5.6)$$

From (5.6) it follows that $A = (\bar{y} - B\bar{x}) + P\underset{\sim}{\delta}$ and this is the same as (5.5).

It should be noted that the principal components method could have been used in place of the least squares approach. Having crossover data in hand does not preclude its use. Thus $\bar{t}_y = \bar{t}_x$ and $\delta = 0$ in (5.5). We can modify the objective function Q of (2.11) and get a perfect fit, i.e., because of (5.2),

$$0 = \|p - Bq\|^2 = p^Tp - 2p^TBq + q^Tq$$

$$= 2 - 2\ \text{Trace}\{pq^TB^T\} \qquad (5.7)$$

and the Trace factor of (5.7) is unity.

With the above in mind let us turn to the question of extending the principal components method to several arrays and several crossover regions. It is simpler if we have crossover data so let's describe that situation first.

The notation is the same as in Section 4. Let $q_i(s)$ be the first principal component of $D_{ii}(s)$ (see (4.6)). The orientation changes $\{B_i\}$ will be selected first by minimizing

$$Q = \sum_{i<j} \sum_{s} \| q_i^T(s)B_i - q_j^T(s)B_j \|^2$$

$$= \sum_{i<j} \sum_{s} \{2 - 2 \text{ Trace}[q_i(s)q_j^T(s)B_j^T B_i]\} \qquad (5.8)$$

and the summations are for $1 \leq i < j \leq K$ and $s \in C_i \cap C_j$. As before, this is tantamount to maximizing

$$W = \sum_{i<j} \sum \text{Trace}\{D_{ij}B_j^T B_i\} \quad \text{with} \quad D_{ij} = \sum_{s} q_i(s)q_j^T(s) \qquad (5.9)$$

Since this has the same structure as (4.8), the same solution algorithm can be used. Also the system (4.17) and (4.18) can be used to obtain the $\{A_i\}$.

The procedure changes a bit when the data are not of the crossover form. That is, when we are splicing together segments of track and there are no overlaps. One might liken this problem as being similar to putting in water pipes in a large building. Several subcontractors have put in their pipes and our job is to connect it all up. The pieces must be moved so that the ends butt up against one another.

With this view, we need a change in terminology. Let $S_1,\ldots,S_R$ represent the crossover points, or connection points. To each S_s there corresponds two arrays, i.e., an unique (i,j)

pair. Let $T_i(s)$ and $T_j(s)$ be the point counts that produce the data. It is assumed that they do not overlap. Each is sufficiently extensive so as to provide stable estimates for the covariance matrices $D_{ii}(s)$ and $D_{jj}(s)$, yet not so extensive as to compromise the assumption that the target is moving in a straight line while in the point counts neighboring the connection. This done, we can input equation (5.9) and use the algorithm of Section 4 to estimate the $\{B_i\}$.

It remains to estimate the $\{A_i\}$. To do this the technique preceeding equation (5.5) needs to be expanded. With more than two arrays involved, the objective function Q of (5.8) will not be zero. This means that our reoriented segments of track will not share an exact common line at the connection points. With this possibility in mind we introduce some further quantities.

To each of the sets $T_i(s)$ and $T_j(s)$ adjoin (if necessary) a common time value (or pseudo point count) $t_{ij}(s)$ which will serve as the connecting time. Let $\bar{t}_i(s)$ and $\bar{t}_j(s)$ be the average values of $T_i(s)$ and $T_j(s)$ respectively. For convenience it is assumed that the former preceeds the latter; $\bar{t}_i(s) < \bar{t}_j(s)$. The corresponding pre-adjusted positives are $\bar{x}_i(s)$ and $\bar{x}_j(s)$. Estimate the target's speed in the same array as before and use it to compute the distances:

$$\delta_i(s) = \text{distance traveled from } \bar{t}_i(s) \text{ to } t_{ij}(s)$$

$$\delta_j(s) = \text{distance traveled from } t_{ij}(s) \text{ to } \bar{t}_j(s).$$

These will be used in our immediate goal, namely, to take the linearly adjusted pieces of track

$$\begin{aligned} \hat{x}_i(t;s) &= A_i + B_i x_i(t;s) \\ \hat{x}_j(t;s) &= A_j + B_j x_j(t;s) \end{aligned} \tag{5.10}$$

and model the differences $A_j - A_i$ in terms of the $\delta_i(s)$, $\delta_j(s)$ and other known quantities.

In continuing, let us drop the argument s from the notation. Again we can use the eigenvectors, q_i, to project the track to an axis so we can measure the distances. Specifically, write

$$\begin{aligned} q_i^T \hat{x}_i(t_{ij}) - q_i^T [A_i + B_i \bar{x}_i] &= \delta_i \\ q_j^T [A_j + B_j \bar{x}_j] - q_j^T \hat{x}_j(t_{ij}) &= \delta_j \end{aligned} \tag{5.11}$$

Although we don't know how to compute $\hat{x}_i(t_{ij})$ and $\hat{x}_j(t_{ij})$ they both represent the position at the connecting point, $\hat{x}_i(t_{ij}) = \hat{x}_j(t_{ij})$. Next rewrite (5.11) as

$$\begin{aligned} q_j^T A_j &= q_j^T \hat{x}_i(t_{ij}) - q_j^T B_j \bar{x}_j + \delta_j \\ q_i^T A_i &= q_i^T \hat{x}_i(t_{ij}) - q_i^T B_i \bar{x}_i - \delta_i \end{aligned} \tag{5.12}$$

Then apply the same technique as used in the proof of (5.5). The result is

$$A_j = \hat{x}_j(t_{ij}) - B_j\bar{x}_j + q_j\delta_j$$

$$A_i = \hat{x}_i(t_{ij}) - B_i\bar{x}_i - q_i\delta_i \tag{5.13}$$

The unknown quantities $\hat{x}_i(t_{ij})$ will cancel when we take differences. Let us restore the s to the notation and record the result.

$$A_j - A_i = B_i\bar{x}_i(s) - B_j\bar{x}_j(s) + \delta_j(s)q_j(s) + \delta_i(s)q_i(s) \tag{5.14}$$

The system (5.14) is overdetermined and has no solution. Let us develop the least squares compromise. To shorten the writing let $g_{ij}(s)$ represent the right hand side of (5.14). Many details will be omitted because we follow the pattern at the end of Section 4.

Let

$$Q = \sum_{i<j} \sum_{s} \| A_j - A_i - g_{ij}(s) \|^2 \tag{5.15}$$

and

$$Q_r = \sum_{i<r} \sum_{s} \| A_r - A_i - g_{ij}(s) \|^2 + \sum_{j>r} \sum_{s} \| A_j - A_r - g_{rj}(s) \|^2$$

Then

$$\frac{\partial Q}{\partial A_r} = \sum_{i<r} \sum_{s} \{A_r - A_i - g_{ir}(s)\}$$

$$- \sum_{j>r} \sum_{s} \{A_j - A_r - g_{rj}(s)\} \qquad (5.16)$$

which is set equal to zero for each $r = 1,\ldots,K$. Let n_{ij} be the number of (i,j) connections that exist, i.e., $n_{ij} = \sum_{s} 1$, and proceed. For $r = 1,\ldots,K$ we have the system

$$- \sum_{i<r} n_{ir}A_i + A_r\{\sum_{i<r} n_{ir} + \sum_{j>r} n_{rj}\} - \sum_{j>r} n_{rj}A_j$$

$$= \sum_{i<r} \sum_{s} g_{ir}(s) - \sum_{j>r} \sum_{s} g_{rj}(s) \qquad (5.17)$$

and the $g_{ir}(s)$ and $g_{rj}(s)$ can be found from the right side of (5.14). As before, the columns of the coefficient matrix add to zero and the rank of the system is K-1. We set $A_1 = 0$ (i.e., the location of the first array is assumed known) and solve.

REFERENCES

1. T.W. Anderson, An Introduction to Multivariate Statistical Analysis. New York: John Wiley & Sons, 1958.

2. R.A. Frazer, W.J. Duncan, A.R. Collar, Elementary Matrices. New York: MacMillan, 1947.

3. F.A. Graybill, An Introduction to Linear Statistical Models. New York: McGraw Hill, 1961.

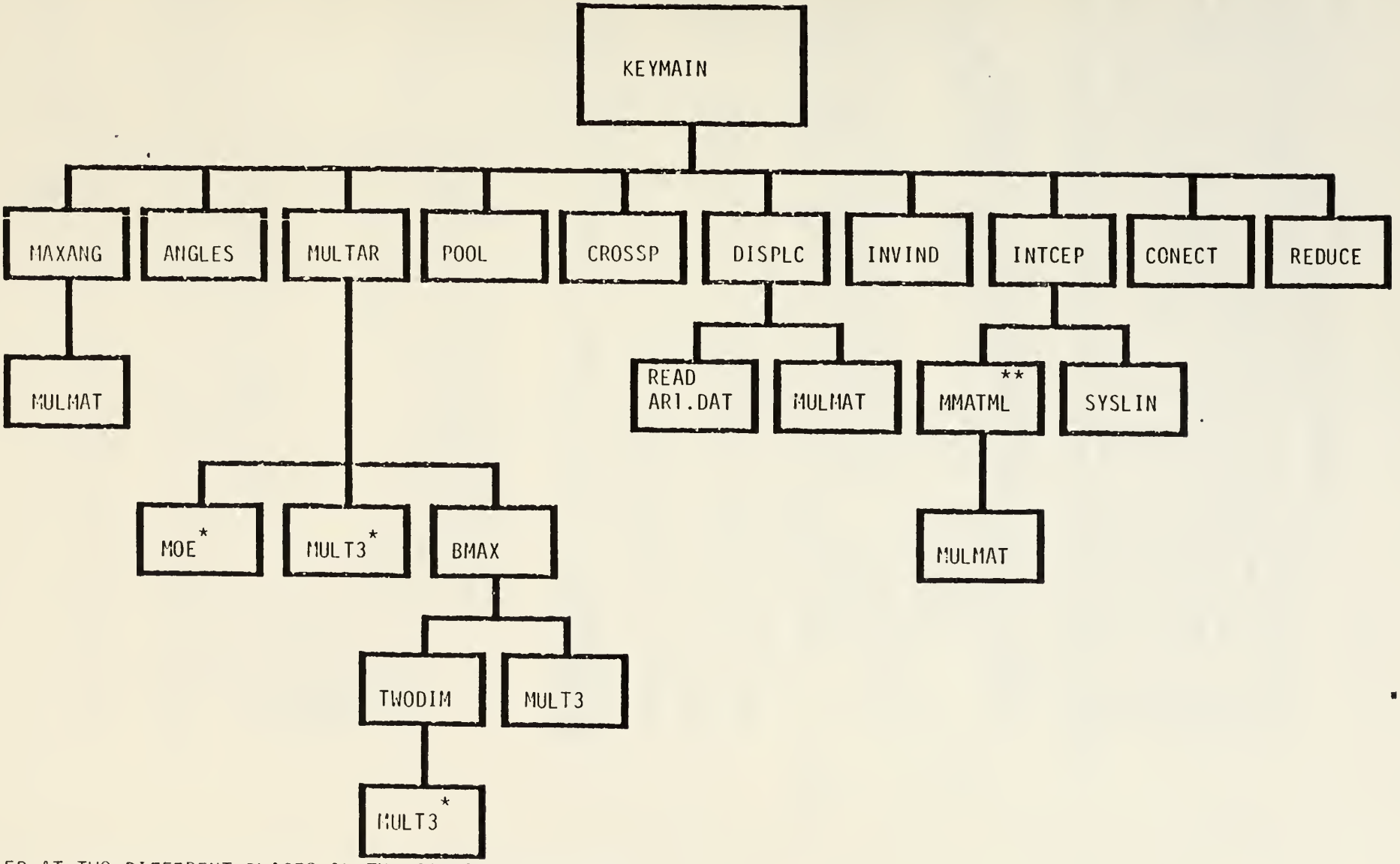
4. G. Gygax, "The Simulation of Remotely Measured Paths of Underwater Vehicles for the Purpose of Monitoring the Calibration of Test Ranges," Master's Thesis, USN Post-graduate School, September 1985.

5. D.W. Marquardt, "An Algorithm for Least Square Estimation of Nonlinear Parameters," Journal of the Society of Industrial and Applied Mathematics, Vol. 11, No. 2, June, 1963.

6. R. Read, "Analysis of Tracking Data," USNPS Technical Paper, NPS55-83-031PR, October 1983.

APPENDIX

SUBROUTINE CALL CHART



*CALLED AT TWO DIFFERENT PLACES IN THE CALLING SUBROUTINE
**CALLED AT FIVE DIFFERENT PLACES IN THE CALLING SUBROUTINE

```
C                          PROGRAM KEYMAIN
C
C     ****************************************************************
C     ***  This serves as a main program to call the subroutines   ***
C     ***  used to estimate the sensor array displacements and     ***
C     ***  the array re-orientation angles for the  Keyport range  ***
C     ***  calibration project.  It reads in data from a disk      ***
C     ***  file specified by the user. It prompts the user for     ***
C     ***  this data and for sensor array I.D. information.        ***
C     ***  -- 26 August 1985                                       ***
C     ****************************************************************
C
C     ...  Declare all integer variables:
      INTEGER*4 IND2(2,30), IND1(30,30), R1, NUMREC, I, J, IDL,
     2          K, IA(30), TESTC, NS1(30), IND(30,30), R, NS(30),
     3          FF(30,30), IK, KM, IDR, IND2R(2,30), DATSET(30),
     4          CHOICE, OUT, IN
C
C     ...  Declare all real variables in double precision:
      REAL*8 AA(200,6), CROSSA(30,3,3), MEAN(30,6), DEV(30,3,3),
     2          XB(30,6), EP, B(30,3,3), BB(3,3), XBB(30,6), A(30,3),
     3          DEL(30,3), D(30), P(30), C(3,3), RM(6), PP, EA(30,3)
C
C     ...  Declare the character variables:
      CHARACTER DSNAME*13, ANSWER*3
C
      LOGICAL EXST
C
      PARAMETER(OUT=6,IN=5)
C
      WRITE(OUT,*) ' This is a user friendly program. Hello !!. '
      WRITE(OUT,*) '    '
      R1 = 0
C
C     ... The IND2 and IND2R matrices serve to store array pair
C             identification information and relate it to the input
C             (six column) crossover data sets.
C     ... Initialize IND2 and IND2R matrices to zero:
      DO 30 I = 1,2
      DO 30 J = 1,30
      IND2(I,J) = 0
      IND2R(I,J) = 0
 30   CONTINUE
C
 5    WRITE(OUT,*) ' Please enter the name of the data set on disk: '
C
C     ...  R1 counts the number of data sets input by the user.
      R1 = R1 + 1
C
```

```
      READ(IN,'(A)') DSNAME
C      ... Check to make sure data file exists:
       INQUIRE(FILE=DSNAME,EXIST=EXST)
       IF(.NOT. EXST) THEN
         WRITE(OUT,*)' The file does not exist.'
 9       WRITE(OUT,*)' Do you want to (1) try again or (2) abort?'
         READ(IN,*) CHOICE
         IF(CHOICE .EQ. 1) THEN
             R1 = R1 - 1
             GOTO 5
         ELSE IF(CHOICE .EQ. 2) THEN
             WRITE(OUT,*)' Program terminating at your request.'
             STOP
         ELSE IF(CHOICE .NE. 1 .AND. CHOICE .NE. 2) THEN
             WRITE(OUT,*)' Please enter integer 1 or 2 .'
             GOTO 9
         END IF
       END IF
C
C
       OPEN(1,FILE=DSNAME,STATUS='OLD')
C
C      ... Read data from file and count number of records (NUMREC):
C
       NUMREC = 0
 10    NUMREC = NUMREC + 1
       READ(1,*,END=20,ERR=15)(AA(NUMREC,J),J=1,6)
       GOTO 10
C
C      ...  Notify operator of file read error:
 15      WRITE(OUT,*)'    '
         WRITE(OUT,*)' An error was detected in reading the file,'
         WRITE(OUT,*)'      but execution will continue anyway ...'
         WRITE(OUT,*)'    '
         WRITE(OUT,*)'    '
C
 20    NUMREC = NUMREC - 1
       WRITE(OUT,*) ' There are ',NUMREC,' records in data set ',DSNAME
       CLOSE(UNIT=1)
C      ... The number of records per input data set are accumulated
C              in the vector NS1.
       NS1(R1) = NUMREC
C
C      ... Define left & right sensor no.'s for matrix IND2:
C              "left" will be indexed with a one and "right" will
C              be indexed with a two.
       WRITE(OUT,*) ' Name your left sensor array number: '
       READ(IN,*) IDL
       WRITE(OUT,*) ' Name your right sensor array number: '
       READ(IN,*) IDR
C
C      ... The left and right array ID's are entered in the
C              first and second (respectively) rows of IND2.
       IND2(1,R1) = IDL
       IND2(2,R1) = IDR
```

```
C       ... Begin subroutine calls:
C               Subroutine CROSSP takes each input data set AA, which
C               has NUMREC records, and returns RM -the six component m
C               vector and C-the three by three matrix of sums of cross
C               deviations from the means.  These are computed sequenti
C               and stored in the arrays MEAN and CROSSA as they are co
C
        CALL CROSSP(NUMREC,AA,RM,C)
        DO 50 I = 1,3
        DO 50 J = 1,3
        CROSSA(R1,I,J) = C(I,J)
 50     CONTINUE
        DO 60 KM = 1,6
        MEAN(R1,KM) = RM(KM)
 60     CONTINUE
C
C
        WRITE(OUT,*) ' Would you like to call another data set ? '
        READ(IN,'(A)') ANSWER
        IF (ANSWER .EQ. 'YES' .OR. ANSWER .EQ. 'Y') GOTO 5
        IF (ANSWER .EQ. 'yes' .OR. ANSWER .EQ. 'y') GOTO 5
C
C       ...  The subroutine CONECT (Gygax) takes as input the number
C                 of data sets R1, and the left/right array ID matrix
C                 IND2 and determines whether the problem input by the
C                 user is connected. That is, all sensor arrays in the
C                 problem communicate with one another via a string of
C                 crossover data sets.  If this test fails, then the
C                 output variable TESTC is zero and the user is prompt
C                 to use one of the three options listed in the WRITE
C                 statements below.
C
C       ...  If the problem is connected, then TESTC is not zero and
C            the subroutine returns a value for:
C                 K, the number of sensor arrays in the problem.
C                 IND1, a useful conversion of IND2.
C                 IA, the list of sensor ID's in the order
C                     maintained by the program.
C                 IND2R, the submatrix of IND2 that represents data se
C                     connected to the first data set.
C                 DATSET, the number of crossover data sets connected
C                     to the first input data set.
        CALL CONECT(OUT,R1,IND2,K,IND1,IA,TESTC,IND2R,DATSET)
        IF (TESTC .EQ. 0) THEN
           WRITE(OUT,*) 'All of your arrays are not connected. '
           WRITE(OUT,*) 'Do you want to :'
           WRITE(OUT,*) '   (1) Quit now'
           WRITE(OUT,*) '   (2) Input more data'
           WRITE(OUT,*) '   (3) Continue, using the first connected set
 65        WRITE(OUT,*) 'Please enter the number of your choice '
           READ(IN,*) I
           IF (.NOT. ((I .EQ. 1).OR.(I .EQ. 2).OR.(I .EQ. 3))) GOTO 65
           GOTO (66, 67, 68),I
 66        WRITE(OUT,*) 'Program terminating. Adios, Amigo !'
           STOP
```

```
 67         GOTO 5
C
C           ... The subroutine REDUCE (Gygax) does nothing unless option 3
C                   above is evoked.  In that case it modifies the
C                   variable CROSSA, MEAN, R1, K, IND1, and IA so that
C                   they contain only the information required by the
C                   connected problem called for by this option.
C
 68         CALL REDUCE(CROSSA,MEAN,R1,K,IND1,IA,IND2R,DATSET)
        END IF
C
C       ... Feed the arrays output by CONECT or REDUCE into POOL:
C               The subroutine POOL checks on the number of data sets
C               associated with each pair of sensor arrays.  If this
C               value is one or zero it does nothing.  Otherwise it
C               (in effect) pools all the data associated with each
C               unique sensor array pair into one data set (using the
C               within groups sum of squares technique for crossproducts,
C               and the weighted average technique for means.  Thus,
C               CROSSA is converted to DEV; MEAN to XB; R1 to R; IND1 to
C               IND; and NS1 to NS.  The first six called variables are
C               input and the last five are output.
C
        CALL POOL(K,R1,IND1,CROSSA,MEAN,NS1,IND,DEV,XB,R,NS)
C
C       ... Array IND output by POOL goes into INVIND:
C               The subroutine INVIND takes the variables K, R, and IND
C               and outputs the K by K matrix FF.  This is an upper
C               triangular incidence matrix which contains a one in
C               the position i,j (i<j) only if there exists a set of
C               crossover data involving both the i-th and j-th
C               sensors (as indexed in IA).  FF is essentially an
C               inverse of IND.
        CALL INVIND(K,R,IND,FF)
C
C       ... Array FF is output from INVIND and input to MULTAR.
C       ... Set epsilon, the tolerable error level used in the
C               stopping rule of the subroutine MULTAR:
        EP = 1.D-6
C
C       ... The subroutine MULTAR computes, by iteration, the K
C               orthonormal re-orientation matrices for each sensor
C               listed in IA.  Since these are all relative to the
C               first sensor, the first matrix in the output B is
C               the 3 x 3 identity matrix.
C
        CALL MULTAR(EP,FF,DEV,   K,R,B)
C
C       ... Array B is output from MULTAR and input to INTCEP:
        CALL INTCEP(FF,NS,B,XB,K,R,A)
```

```
C       ... Array A is output from INTCEP and input to DISPLC:
C       ... The subroutine DISPLC takes variables K, B, A, and IA as
C               input, reads the sensor array location file and comput
C               the estimated displacement of each of the K sensors in
C               the problem.  The output DEL is the set of displacemen
C               vectors and D is the corresponding set of K displaceme
C               distances.  Since these are all relative to the first
C               sensor, the first row of DEL and first element of D ar
C               zero.
C
        CALL DISPLC(K,B,A,IA,DEL,D)
C
C       ... Prepare the sequential computation of the maximum angles
C             of sensor rotation.
C
        DO 75 IK=1,K
        DO 70 I=1,3
        DO 70 J=1,3
 70     BB(I,J) = B(IK,I,J)
C       ... Array BB is input to MAXANG
C
C       ... The subroutine MAXANG takes a 3 by 3 orthonormal matrix BB
C               and computes PP, the largest possible rotation angle
C               experienced by any vector transformed by BB.
C
        CALL MAXANG(BB,PP)
        P(IK) = PP
 75     CONTINUE
C
C       ... The subroutine ANGLES converts each of the K orthonormal
C               matrices in B to their representation as the three
C               Euler angles:  Roll, Pitch, and Yaw.  Since these are
C               relative to the first sensor, the first row of this
C               matrix is zero.
C
        CALL ANGLES(K,B,EA)
C
C
C       ... Prepare to write the output to files:
        OPEN(3,FILE='KEYFIL1.DAT',STATUS='OLD')
        OPEN(4,FILE='KEYFIL2.DAT',STATUS='OLD')
        OPEN(7,FILE='KEYFIL3.DAT',STATUS='OLD')
        OPEN(8,FILE='KEYFIL4.DAT',STATUS='OLD')
C
        WRITE(OUT,*) ' Displacement and rotation (magnitude):'
        DO 77 IK = 1,K
C       ... Write the vectors IA, D and P to FILE1
        WRITE(3,85) IA(IK), D(IK), P(IK)
        WRITE(OUT,85) IA(IK), D(IK), P(IK)
C
C       ... Write the vector IA and matrices A and DEL to FILE2.
        WRITE(4,80) IA(IK), (DEL(IK,L),L=1,3), (EA(IK,LK),LK=1,3)
 77     CONTINUE
```

```
      WRITE(OUT,*) ' Displacement vectors and Euler angles :'
      DO 78 IK = 1,K
      WRITE(OUT,80) IA(IK), (DEL(IK,L),L=1,3), (EA(IK,LK),LK=1,3)
 78   CONTINUE
 80   FORMAT(1X,I5,1X,3F12.4,1X,3(1X,F10.7))
 85   FORMAT(2X,I5,2X,F10.2,2X,F10.6)
C
C     ... Write the three dimensional array B to FILE3.
      WRITE(OUT,*)'  '
      WRITE(OUT,*)' Array B output by KEYMAIN: '
      DO 90 IK = 1,K
      WRITE(OUT,*)' '
      WRITE(OUT,*)' '
      DO 90 I = 1,3
      WRITE(7,100)(B(IK,I,J),J=1,3)
      WRITE(OUT,100)(B(IK,I,J),J=1,3)
 90   CONTINUE
 100  FORMAT(2X,3F15.10)
C
      DO 105 IK = 1,K
      WRITE(8,115)(A(IK,J),J=1,3)
 105  CONTINUE
 115  FORMAT(2X,3F15.5)
C
      WRITE(OUT,110)
 110  FORMAT(//,' Program terminating.  Operation complete.  Bye. ')
C
      CLOSE(UNIT=3)
      CLOSE(UNIT=4)
      CLOSE(UNIT=7)
      CLOSE(UNIT=8)
C
C
      STOP
      END



      SUBROUTINE CROSSP (N,AA,MEAN,CROSSA)
C
C     ***************************************************************
C     *** THIS PROGRAM CALCULATES THE CROSSPRODUCT DEVIATIONS FOR     **
C     *** THE RAW DATA ARRAY AA (A MATRIX FOR EACH CROSSOVER SET)     **
C     *** THE SET OF MEANS IS DEVELOPED AND LABELED "MEAN"            **
C     *** CROSSA IS THE SUM OF CROSSPRODUCT DEVIATIONS FROM THE MEANS **
C     ***************************************************************
C
      INTEGER*4 N
      REAL*8 AA(200,6), MEAN(6), CROSSA(3,3), SUM
      REAL*8 SUMXX, SUMXY, SUMXZ, SUMYX, SUMYY, SUMYZ, SUMZX, SUMZY
      REAL*8  SUMZZ, XX, XY, XZ, YX, YY, YZ, ZX, ZY, ZZ
C
C      ... N is the number of rows in the data matrix AA
C      ... Compute the vector of means.
```

```
        DO 20 J = 1,6
        SUM = 0.D0
        DO 10 I = 1,N
        SUM = SUM + AA(I,J)
 10     CONTINUE
        MEAN(J) = SUM / DBLE(N)
 20     CONTINUE
C
C
C       ... Begin the crossproduct operations:
C           Initialize the nine sums to zero.
      SUMXX=0.D0
      SUMXY=0.D0
      SUMXZ=0.D0
      SUMYX=0.D0
      SUMYY=0.D0
      SUMYZ=0.D0
      SUMZX=0.D0
      SUMZY=0.D0
      SUMZZ=0.D0
C
      DO 100 IA = 1,N
C
C     ... Create the crossproduct deviations for row IA of matrix AA:
C
      XX=(AA(IA,1)-MEAN(1))*(AA(IA,4)-MEAN(4))
      XY=(AA(IA,1)-MEAN(1))*(AA(IA,5)-MEAN(5))
      XZ=(AA(IA,1)-MEAN(1))*(AA(IA,6)-MEAN(6))
      YX=(AA(IA,2)-MEAN(2))*(AA(IA,4)-MEAN(4))
      YY=(AA(IA,2)-MEAN(2))*(AA(IA,5)-MEAN(5))
      YZ=(AA(IA,2)-MEAN(2))*(AA(IA,6)-MEAN(6))
      ZX=(AA(IA,3)-MEAN(3))*(AA(IA,4)-MEAN(4))
      ZY=(AA(IA,3)-MEAN(3))*(AA(IA,5)-MEAN(5))
      ZZ=(AA(IA,3)-MEAN(3))*(AA(IA,6)-MEAN(6))
C
C     ... Accumulate the sums:
      SUMXX=XX+SUMXX
      SUMXY=XY+SUMXY
      SUMXZ=XZ+SUMXZ
      SUMYX=YX+SUMYX
      SUMYY=YY+SUMYY
      SUMYZ=YZ+SUMYZ
      SUMZX=ZX+SUMZX
      SUMZY=ZY+SUMZY
      SUMZZ=ZZ+SUMZZ
C
 100  CONTINUE
C
```

```
C     ... Create the CROSSA matrix:
      CROSSA(1,1)=SUMXX
      CROSSA(1,2)=SUMXY
      CROSSA(1,3)=SUMXZ
      CROSSA(2,1)=SUMYX
      CROSSA(2,2)=SUMYY
      CROSSA(2,3)=SUMYZ
      CROSSA(3,1)=SUMZX
      CROSSA(3,2)=SUMZY
      CROSSA(3,3)=SUMZZ
C
      RETURN
C
      END



       SUBROUTINE CONECT (OUT,R1,IND2,K,IND1,IA,TESTC,IND2R,DATSET)
C
C      ***************************************************************
C      *** This subroutine checks for the connectedness of the     ***
C      *** input data sets. If the problem is connected then the   ***
C      *** user is informed and the array pairs are printed on the ***
C      *** screen; if not connected, then the user is prompted to  ***
C      *** select one of three options - quit, add conecting data  ***
C      *** sets, or run the program using the first connected set  ***
C      *** that was input.            Gygax - July 1985            ***
C      ***************************************************************
C
C
C      ...Variable declarations.
C
      INTEGER*4 R1,K,IND2(2,30),IND1(30,30),I,J,IA(30),FIRST
      INTEGER*4 LIST(30),BEGIN,HALT,DISCON,L,M,O,TESTC,IND2R(2,30)
      INTEGER*4 DATSET(30),COUNT,SAVE(2,30), OUT
C
C      ...Initialize the values of FIRST and COUNT:
C
      FIRST = 0
      COUNT = 0
C
C      ...Make vector IA = list of all arrays (w/o repeats) in IND2
C         and get the value for K = # of individual arrays.
C
      IA(1) = IND2(1,1)
      IA(2) = IND2(2,1)
      K = 3
      IF (R1 .EQ. 1) GOTO 60
      DO 50 I = 1,R1
         DO 40 J = 1,2
            M = K - 1
            DO 30 L = 1,M
               IF (IND2(J,I) .EQ. IA(L)) GOTO 40
 30         CONTINUE
```

```
            IA(K) = IND2(J,I)
            K = K + 1
 40       CONTINUE
 50       CONTINUE
 60    K = K - 1
C
       WRITE(OUT,*)'R1  ',R1
       WRITE(OUT,*)'K   ',K
C      ... For each column of IND1 (columns correspond to data sets) th
C          entries are all zero except for the row that corresponds to
C          the left array (= 1) and the right array (= 2).
C
      DO 80 I = 1,R1
         DO 70 J = 1,K
            IND1(J,I) = 0
            IF (IND2(1,I) .EQ. IA(J)) IND1(J,I) = 1
            IF (IND2(2,I) .EQ. IA(J)) IND1(J,I) = 2
 70       CONTINUE
 80    CONTINUE
C
C      ...Check to see if all the arrays are connected.
C
      TESTC = 1
      LIST(1) = -IA(1)
      DO 131 I = 1,R1
         IF (IND2(1,I) .EQ. -LIST(1)) IND2(1,I) = -IND2(1,I)
         IF (IND2(2,I) .EQ. -LIST(1)) IND2(2,I) = -IND2(2,I)
 131  CONTINUE
      BEGIN = 1
      HALT = 1
 140  IF (.NOT. (BEGIN .LE. HALT)) GOTO 170
      NODE = LIST(BEGIN)
      BEGIN = BEGIN + 1
      DO 150 I = 1,R1
         IF (.NOT.((NODE.EQ.IND2(1,I)).AND.(IND2(2,I).GT.0))) GOTO 150
            HALT = HALT + 1
            LIST(HALT) = -IND2(2,I)
            DO 141 J = 1,R1
               IF (IND2(1,J) .EQ. -LIST(HALT)) IND2(1,J) = -IND2(1,J)
               IF (IND2(2,J) .EQ. -LIST(HALT)) IND2(2,J) = -IND2(2,J)
 141        CONTINUE
 150  CONTINUE
      DO 160 I = 1,R1
         IF (.NOT.((NODE.EQ.IND2(2,I)).AND.(IND2(1,I).GT.0))) GOTO 160
            HALT = HALT + 1
            LIST(HALT) = -IND2(1,I)
            DO 151 J = 1,R1
               IF (IND2(1,J) .EQ. -LIST(HALT)) IND2(1,J) = -IND2(1,J)
               IF (IND2(2,J) .EQ. -LIST(HALT)) IND2(2,J) = -IND2(2,J)
 151        CONTINUE
 160  CONTINUE
      GOTO 140
 170  CONTINUE
```

```
C       ...Print out the matched pairs.
        DISCON = 0
        WRITE(OUT,230)
        DO 200 I = 1,R1
        IF (IND2(1,I) .LT. 0) GOTO 190
        IF (IND2(1,I) .EQ. 0) GOTO 200
        IF ((IND2(1,I) .GT. 0) .AND. (DISCON .EQ. 1)) GOTO 200
        FIRST = FIRST + 1
        DISCON = 1
        TESTC = 0
        BEGIN = 1
        HALT = 1
        LIST(1) = IND2(1,I)
        GOTO 200
  190   WRITE(OUT,240) -IND2(1,I),-IND2(2,I)
        IF ((FIRST.EQ.0).OR.((FIRST.EQ.1).AND.(DISCON.EQ.1))) THEN
           COUNT = COUNT + 1
           IND2R(1,COUNT) = -IND2(1,I)
           IND2R(2,COUNT) = -IND2(2,I)
           DATSET(COUNT) = I
        END IF
        SAVE(1,I) = -IND2(1,I)
        SAVE(2,I) = -IND2(2,I)
        IND2(1,I) = 0
        IND2(2,I) = 0
  200   CONTINUE
        IF (DISCON .EQ. 1) GOTO 140
        DO 220 I = 1,R1
           IND2(1,I) = SAVE(1,I)
           IND2(2,I) = SAVE(2,I)
  220   CONTINUE
        RETURN
  230   FORMAT(1X,'THE FOLLOWING PAIRS ARE CONNECTED :')
  240   FORMAT(1X,14I5)
        END



        SUBROUTINE REDUCE (CROSSA,MEAN,R1,K,IND1,IA,IND2R,DATSET)
C
C       ***************************************************************
C       *** This is a specialized subroutine that is used  when     ***
C       *** option three is involked as a result of a failed con-   ***
C       *** nectedness test. The disconnected data sets must be     ***
C       *** removed from the variables CROSSA and MEAN, and other   ***
C       *** program supporting variables must be adjusted.          ***
C       *** Gygax - July 1985                                       ***
C       ***************************************************************
C       ... Variable declarations.
C
        INTEGER*4 R1,K,IND1(30,30),IA(30),IND2R(2,30),I,J,L,M,DATSET(30)
C
        REAL*8 CROSSA(30,3,3),MEAN(30,6)
C
```

```
C       ... Compute the new, reduced R1:
        DO 10 I = 1,30
           IF (IND2R(1,I) .EQ. 0) GOTO 20
 10     CONTINUE
 20     R1 = I - 1
C
C       ... Make a new, reduced vector IA = list of all arrays in
C           IND2R w/o repeats. Also, compute a new K.
C
        IA(1) = IND2R(1,1)
        IA(2) = IND2R(2,1)
        K = 3
        IF (R1 .EQ. 1) GOTO 60
        DO 50 I = 1,R1
           DO 40 J = 1,2
              M = K - 1
              DO 30 L = 1,M
                 IF (IND2R(J,I) .EQ. IA(L)) GOTO 40
 30           CONTINUE
              IA(K) = IND2R(J,I)
              K = K + 1
 40        CONTINUE
 50     CONTINUE
 60     K = K - 1
C
C       ... Remake the reduced matrix IND1 - for each column in IND1
C           (corresponding to a data set) the entries are zero except
C           for the enties corresponding to the left array (= 1) and
C           right array (= 2).
C
        DO 80 I = 1,R1
           DO 70 J = 1,K
              IND1(J,I) = 0
              IF (IND2R(1,I) .EQ. IA(J)) IND1(J,I) = 1
              IF (IND2R(2,I) .EQ. IA(J)) IND1(J,I) = 2
 70        CONTINUE
 80     CONTINUE
C
C       ... Reduce the arrays CROSSA and MEAN to account for the
C           removed data sets.
C
        DO 120 I = 1,R1
           DO 90 J = 1,6
              MEAN(I,J) = MEAN(DATSET(I),J)
 90        CONTINUE
           DO 110 J = 1,3
              DO 100 L = 1,3
                 CROSSA(I,J,L) = CROSSA(DATSET(I),J,L)
 100          CONTINUE
 110       CONTINUE
 120    CONTINUE
        RETURN
        END
```

```
      SUBROUTINE POOL(K,R1,IND1,CROSSA,MEAN,NS1,IND,DEV,XB,R,NS)
C
C     ****************************************************************
C     ***  Pools the means and the cross product deviations        ***
C     ***  so that each cross covariance corresponds to a unique   ***
C     ***  sensor array pair.                                      ***
C     ****************************************************************
C
      INTEGER*4 K,R1,IND1(30,30),IND(30,30),R,NS1(30),NS(30),IC,IS
      REAL*8 CROSSA(30,3,3), MEAN(30,6), DEV(30,3,3), XB(30,6),
     +     TX(6), DD(3,3),RNS, DNS1
C
C
C     ... Outputs replacement variables IND, DEV, XB, R, and NS
C         for use in MULTAR and INTCEP. That is:
C         IND replaces IND1, DEV is the pooled version of CROSSA,
C         XD is the pooled version of MEAN, R replaces R1,
C         and NS replaces NS1.
C
C     ... Identify data subsets in IND1 which are in the wrong order
C         for use in the algorithms and transpose them.
      DO 20 IR=1,R1
      DO 10 I=1,K
      IF (IND1(I,IR) .EQ. 0) THEN
                  GOTO 5
         ELSE IF (IND1(I,IR) .EQ. 1) THEN
                  IK=I
         ELSE IF (IND1(I,IR) .EQ. 2) THEN
                  JK=I
      END IF
    5 CONTINUE
   10 CONTINUE
      IF (IK .LT. JK) GOTO 20
      DO 12 I=1,3
      IP3=I+3
      TX(I)=MEAN(IR,IP3)
      TX(IP3)=MEAN(IR,I)
      DO 12 J=1,3
   12 DD(I,J) = CROSSA(IR,J,I)
      DO 15 I=1,3
      IP3=I+3
      MEAN(IR,I)=TX(I)
      MEAN(IR,IP3)=TX(IP3)
      DO 15 J=1,3
   15 CROSSA(IR,I,J)=DD(I,J)
   20 CONTINUE
C     ... The transpositions are completed.
C     ... Locate the subsets to be pocled and do the pooling.
C         Initialize the variables.
      IC = 1
C     ... Zero the IND array.
      DO 25 IT=1,K
      DO 25 IR=1,R1
      IND(IT,IR)=0
   25 CONTINUE
```

```
C     ... Start the algorithm.
      KM1=K-1
      DO 40 I=1,KM1
      IP1 = I + 1
      DO 40 J=IP1,K
      IS = 0
C     ... Zero the arrays initially.
      NS(IC) = 0
      DO 28 IT=1,3
      ITP3=IT+3
      TX(IT)=0.0D+0
      TX(ITP3)=0.0D+0
      DO 28 JT=1,3
 28   DEV(IC,IT,JT)=0.0D+0
C     ... Initialization is completed for this I,J pair.
C
      DO 35 IR=1,R1
      IF (IND1(I,IR) .EQ. 0 .OR. IND1(J,IR) .EQ. 0) GOTO 35
      DNS1 = DBLE(NS1(IR))
C     ... Set a flag to show a viable I,J pair:
      IS=1
C     ... Accumulate weighted sum of means:
      DO 32 IT=1,6
      TX(IT) = TX(IT) + DNS1 * MEAN(IR,IT)
 32   CONTINUE
C     ... Accumulate sums of crossproducts.
      DO 33 IT=1,3
      DO 33 JT=1,3
      DEV(IC,IT,JT) = DEV(IC,IT,JT) + CROSSA(IR,IT,JT)
 33   CONTINUE
C     ... Accumulate sample sizes:
      NS(IC) = NS(IC) + NS1(IR)
 35   CONTINUE
      IF (IS .EQ. 0) GOTO 40
C     ... Convert pooled crossproducts into cross covariances.
      RNS = DBLE(NS(IC))
      DO 36 IT=1,3
      DO 36 JT=1,3
 36   DEV(IC,IT,JT)=DEV(IC,IT,JT)/RNS
C     ... Convert weighted sums into pooled means:
      DO 38 IT=1,6
 38   XB(IC,IT) = TX(IT) / RNS
C     ... Set the values in the IND matrix and update the counter:
      IND(I,IC) = 1
      IND(J,IC) = 2
      IC = IC + 1
 40   CONTINUE
C     ... When finished, the counter needs correction.
      R=IC - 1
      RETURN
      END
```

```
      SUBROUTINE INVIND(K,R,IND,FF)
C
C     ***************************************************************
C     *** Computes the matrix FF from IND                         ***
C     *** FF is used in MULTAR and INTCEP                         ***
C     *** IR indexes the columns of IND, i.e. the crossover pairs.***
C     *** I and J index the sensor arrays.                        ***
C     *** K is the number of arrays in the problem                ***
C     *** R is the number of (array pair) data sets in the problem.***
C     ***************************************************************
C
      INTEGER*4 R, K
      INTEGER*4 IND(30,30), FF(30,30)
C
C     ... Initialize F:
      DO 20 I=1,K
      DO 20 J=1,K
C     ... Change the value of FF to the crossover set index for
C         those array pairs, I<J, that are in the problem.
 20   FF(I,J) = 0
      KM1 = K - 1
      DO 40 I=1,KM1
      IP1 = I + 1
      DO 40 J = IP1,K
      DO 40 IR = 1,R
      IF(IND(I,IR) .EQ. 1 .AND. IND(J,IR) .EQ. 2) FF(I,J) = IR
 40   CONTINUE
C
      RETURN
      END



        SUBROUTINE MULTAR(EP,FF,DEV,   K,R,B)
C
C     ***************************************************************
C     *** This is an iterative program to compute the K orthonormal ***
C     *** matrices in B. They must simultaneously optimize the sum  ***
C     *** of traces objective function.                             ***
C     ***************************************************************
C
C     ---------------------------------------------------------------
C
C     ** K is the number of arrays.
C     ** R is the number of crossover sets.
C     ** EP is the epsilon for stopping the iterations.
C     ** DEV is the set of pooled crosscovariances.
C     ** Calls BMAX, MOE and MULT3.
C     ---------------------------------------------------------------
C
C     ... Declarations
C
      INTEGER*4 R, FF(30,30), K
      REAL*8  B(30,3,3), E(30,3,3), F(30,3,3), DEV(30,3,3)
      REAL*8  EP, REP, W ,W0, BB(3,3), DD(3,3), T(3,3), G(30,3,3)
```

```
C
C       ... Initialize B as K identity matrices.
C
        DO 2 KK=1,K
        DO 2 I = 1,3
        DO 2 J=1,3
        B(KK,I,J) = 0.D0
        IF (I .EQ. J) B(KK,I,J) = 1.D0
 2      CONTINUE
C       ... Initialize the temporary arrays.
 5      DO 10 KK=1,K
        DO 10 I =1,3
        DO 10 J = 1,3
        F(KK,I,J)=0.D0
        E(KK,I,J)=0.D0
10      CONTINUE
C
C       ... Compute W0, the initial value of the objective function.
        CALL MOE(K,FF,DEV,B,W0)
C
C       ... Prepare for the linear transformation of those members of B
C           designated as LEFT with the cross covariances.
        DO 30 KK=2,K
        KKM1=KK-1
        DO 30 JI=1,KKM1
        IR=FF(JI,KK)
        IF (IR .EQ. 0) GOTO 25
        DO 15 IB = 1,3
        DO 15 JB = 1,3
        BB(IB,JB) = B(JI,IB,JB)
        DD(IB,JB) = DEV(IR,IB,JB)
 15     CONTINUE
C
C       ... Perform matrix multiplication and accumulate.
        CALL MULT3(BB,DD,T)
        DO 20 I = 1,3
        DO 20 J = 1,3
 20     E(KK,I,J) = E(KK,I,J) + T(I,J)
 25     CONTINUE
 30     CONTINUE
C
C       ... Prepare for the linear transformations of those members of B
C           designated as RIGHT with the cross covariances.
        KM1 = K-1
        DO 40 KK=1,KM1
        KKP1 = KK + 1
        DO 40 JJ=KKP1,K
        IR=FF(KK,JJ)
        IF (IR .EQ. 0) GOTO 38
        DO 32 IB = 1,3
        DO 32 JB = 1,3
        BB(IB,JB) = B(JJ,IB,JB)
        DD(IB,JB) = DEV(IR,JB,IB)
 32     CONTINUE
C
```

```
C       ... Perform matrix multiplication and accumulate.
        CALL MULT3(BB,DD,T)
        DO 35 I =1,3
        DO 35 J = 1,3
 35     F(KK,I,J)=F(KK,I,J)+T(I,J)
 38     CONTINUE
 40     CONTINUE
C
C       ... Gather the LEFT and RIGHT accumulations.
        DO 50 KK=1,K
        DO 50 I =1,3
        DO 50 J = 1,3
        G(KK,I,J)=E(KK,I,J)+F(KK,I,J)
 50     CONTINUE
C
C       ... Start seeking the maximum on the objective surface.
        DO 60 KK=2,K
        DO 55 I = 1,3
        DO 55 J =1,3
        BB(I,J) = B(KK,I,J)
        DD(I,J) = G(KK,I,J)
  55    CONTINUE
C       ... Subroutine call to optimize an indiviual matrix in B.
        CALL BMAX(EP,DD,BB)
C       ... Insert updated values into the B array.
        DO 58 I=1,3
        DO 58 J=1,3
  58    B(KK,I,J) = BB(I,J)
  60    CONTINUE
C
C       ... Compute new value of the objective function.
        CALL MOE(K,FF,DEV,B,W)
C
C       ... Compare with previous value and decide whether to
C           terminate or iterate.
        REP = EP * W0 / 1.D2
        IF (DABS(W - W0) .GT. REP) GOTO 5
C
        RETURN
        END



        SUBROUTINE MOE(K,FF,DEV,B,W)
C
C       ***************************************************************
C       *** Computes the measure of effectiveness for MULTAR:       ***
C       *** Output is W, the sum of traces of cross covariances     ***
C       *** modified on the left by the B matrix of the left array, ***
C       *** and on the right by the transpose of the B matrix of    ***
C       *** the right array.                                        ***
C       ***************************************************************
C
        INTEGER*4 K, FF(30,30)
        REAL*8 B(30,3,3), DEV(30,3,3), W
```

```
      W=0.D0
      KM1=K-1
      DO 20 IT = 1,KM1
      ITP1=IT+1
      DO 20 JT = ITP1,K
      IR = FF(IT,JT)
      IF (IR .EQ. 0) GOTO 15
      DO 10 I=1,3
      DO 10 M=1,3
      DO 10 J=1,3
      W=W+DEV(IR,I,M)*B(JT,J,M)*B(IT,J,I)
 10   CONTINUE
 15   CONTINUE
 20   CONTINUE
C
      RETURN
      END



      SUBROUTINE BMAX (EP,D,B)
C
C     ****************************************************************
C     *** Iterative program that climbs the surface D*B-TRANS.    ***
C     *** and stops when the change is less than EP times         ***
C     *** previous level divided by 10.  Input D from MULTAR and  ***
C     *** output B which is a 3x3 othonormal matrix.              ***
C     *** Calls the subroutine TWODIM.                            ***
C     *** -- August 1985.                                         ***
C     ****************************************************************
C
      REAL*8 B(3,3), D(3,3), BB(3,3), C(3,3), BM(3,3)
      REAL*8 Q, Q0, EP, R, REP
C     ... Initialize the value on the surface Q for the special
C         case of B = Identity matrix:
      Q = 0.D0
      DO 2 I=1,3
      Q = Q + D(I,I)
 2    CONTINUE
C
C     ... Initialize B as the identity matrix:
      DO 5 I=1,3
      DO 5 J=1,3
      B(I,J) = 0.D0
      IF (I .EQ. J) B(I,J) = 1.D0
 5    CONTINUE
 6    CONTINUE
C
C     ... Compute the intermediate values C = D * B - TRANS :
      DO 25 I = 1,3
      DO 25 J = 1,3
      C(I,J) = 0.D0
      DO 25 L = 1,3
      C(I,J) = C(I,J) + D(I,L) * B(J,L)
 25   CONTINUE
```

```
C
C
C       ... Subroutine call to climb higher on the surface of Q:
        CALL TWODIM(C,BB)
        DO 8 I = 1,3
        DO 8 J = 1,3
        BM(I,J) = B(I,J)
 8      CONTINUE
C
C       ... Replace B with the matrix product.  This updates the B matrix.
        CALL MULT3(BB,BM,B)
C
C       ... Record previous level on the surface:
        Q0 = Q
C
C       ... Compute new value on the surface:
        Q = 0.D0
        DO 10 I = 1,3
        DO 10 J = 1,3
 10     Q = Q + (D(I,J) * B(I,J))
C
C       ... Prepare the stopping rule:
        R = Q - Q0
        REP = EP * Q0 / 10.
C       ... Compares the relative gain with EP; stops if too small:
        IF (R.GT.REP) GOTO 6
C
C
        RETURN
        END



        SUBROUTINE TWODIM (D,B)
C
C       ************************************************************
C       *** FORTRAN subroutine to compute new roll, pitch, and  ***
C       *** yaw matrices (B1,B2,B3) and combine into a single   ***
C       *** orthonormal transformation.  Receives D from BMAX,  ***
C       *** outputs B, calls MULT3.                             ***
C       *** -- 10 June 1985.                                    ***
C       ************************************************************
C
C       ... Dimension variables and declare them to be double precision:
        REAL*8 B(3,3), D(3,3), T(3,3), E(3,3)
        REAL*8 F(3,3), B1(3,3), B2(3,3), B3(3,3)
        REAL*8 S, C, DENOM, SS
C
C       ... Initialize the three matrices.
        DO 5 I=1,3
        DO 5 J=1,3
        B1(I,J)=0.D0
        B2(I,J)=0.D0
        B3(I,J)=0.D0
 5      CONTINUE
```

```
C       ... Develop B1 (roll matrix) from input D:
        S = D(3,2) - D(2,3)
        C = D(2,2) + D(3,3)
        SS = S*S + C*C
        DENOM = DSQRT(SS)
        S = S / DENOM
        C = C / DENOM
C
C       ... Finalize the B1 matrix:
        B1(2,2) = C
        B1(2,3) = S
        B1(3,2) = -S
        B1(3,3) = C
        B1(1,1) = 1.D0
C
C       ... Update the input matrix to adjust for roll:
C
C       ... E = D*B1
        CALL MULT3(D,B1,E)
C
C       ... Develop B2 (pitch matrix) from E:
        S = E(3,1) - E(1,3)
        C = E(3,3) + E(1,1)
        SS = S*S + C*C
        DENOM = DSQRT(SS)
        S = S / DENOM
        C = C / DENOM
C       ... Finalize the B2 matrix:
        B2(1,1) = C
        B2(1,3) = S
        B2(2,2) = 1.D0
        B2(3,1) = -S
        B2(3,3) = C
C       ... Additional update of the input matrix to adjust for pitch:
C       ... F = E*B2
        CALL MULT3(E,B2,F)
C
C       ... Develop B3 (yaw matrix) from F:
C
        S = F(2,1) - F(1,2)
        C = F(1,1) + F(2,2)
        SS = S*S + C*C
        DENOM = DSQRT(SS)
        S = S / DENOM
        C = C / DENOM
C
C       ... Finalize the B3 matrix:
        B3(1,1) = C
        B3(1,2) = S
        B3(2,1) = -S
        B3(2,2) = C
        B3(3,3) = 1.D0
C
```

```
C       ... Final triple multiplication of matrices coupled
C           with transposition:
C           B_transpose = B1 * B2 * B3
        DO 30 I=1,3
        DO 30 J=1,3
        B(J,I) = 0.D0
        DO 30 IK=1,3
        DO 30 KJ=1,3
        B(J,I)=B(J,I)+B1(I,IK)*B2(IK,KJ)*B3(KJ,J)
 30     CONTINUE
        RETURN
        END


      SUBROUTINE MULT3(A,B,C)
C
C     ***************************************************************
C     ***  Multiplies the 3 by 3 matrices A and B to get C        ***
C     ***************************************************************
C
      REAL*8 A(3,3), B(3,3), C(3,3)
C
      DO 10 I=1,3
      DO 10 J=1,3
      C(I,J)= 0.D0
      DO 10 K=1,3
      C(I,J) = C(I,J) + A(I,K)*B(K,J)
 10   CONTINUE
      RETURN
      END



        SUBROUTINE INTCEP(FF,NS,B,XB,K,R,A)
C
C       ***************************************************************
C       *** Sets up and solves the system of linear equations for    ***
C       *** each component of the set of K intercept vectors.       ***
C       *** Requires the output of POOL and MULTAR.                 ***
C       *** -- June 1985.                                           ***
C       ***************************************************************
C
C       ** K is the number of sensor arrays in the problem.
C       ** R is the (pooled) number of crossover data sets.
C       ** NS is the vector of sample sizes for the crossover sets.
C       ** FF is the upper triangular K by K matrix whose values are
C       ** either zero or the index of the data set for the array pair
C       ** identified by the subscripts.
C       ** XB is the R by 6 matrix of means for each of the two arrays
C       ** identified with each of the R data sets.
C       ** B is the K by 3 by 3 collection of orthonormal crientation
C       ** correction matrices returned from MULTAR.
C       ** The output A is the K by 3 matrix of intercept values
C       ** estimated for each of the K arrays in the problem.
C
```

```
C       ... Calls MMATMUL which performs linear transformations.
C       ... Calls SYSLIN which solves systems of linear equations.
C
C       ... Declarations.
C
        INTEGER*4 R, FF(30,30), NS(30), K, I, IR, IP1
        REAL*8  B(30,3,3),XB(30,6),M(30,30),
     1    LHS(30,3),XL(30,3),XLL(30,3),XXL(30,3),XR(30,3),XRR(30,3),
     2    XXR(30,3), E(30,3),F(30,3),A(30,3),AA(30,31),DNS
C
C       ... Begin development of the coefficient matrix M, off diagonal
C           terms.  Also prepare the inputs for the summands on the
C           other side of the equation. Affixes L and R are used to
C           suggest the left and right portions of the expressions.
C
C       ... The values in the coefficient matrix are zero whenever the
C           array pair is not identified with a crossover data set, i.e
C           when FF(i,j)=0 and i is less than j. Otherwise its value is
C           the negative of the sample size for the data set.
C
        KM1=K-1
        DO 10 I=1,KM1
        IP1 = I + 1
        DO 10 J=IP1,K
        IR=FF(I,J)
        IF (IR .EQ. 0) THEN
              M(I,J) = 0.D0
              M(J,I) = 0.D0
              GOTO 10
        END IF
        M(I,J)= -1.D0 * DBLE(NS(IR))
        M(J,I) = M(I,J)
        DO 5 II=1,3
C
C       ...The weighted version requires that the means be multiplied
C          by NS. The first three columns of XB are identified with the
C          left (L) array of the pair; the last 3 with the right (R).
C          This prepares inputs for summands on the other side of the
C          equation.
C
        DNS = DBLE(NS(IR))
        XL(IR,II)=DNS*XB(IR,II)
        IIP3 = 3 + II
 5      XR(IR,II)=DNS*XB(IR,IIP3)
C
 10      CONTINUE
C
C       ... Finish development of the M matrix. The diagonal term in a
C           row is the negative of the total of the other terms in that
C           row. The first column of E is used to accumulate terms.
C
```

```
      DO 15 I=1,K
      M(I,I) = 0.D0
      E(I,1) = 0.D0
      DO 12 J=1,K
 12   E(I,1) = E(I,1) + M(I,J)
      M(I,I) = -1.D0 * E(I,1)
 15   CONTINUE
C
C     ... The coefficient matrix is complete.
C         Next develop the sequence of partial sum arrays for use in
C         the LHS of the system of equations. First initialize.
C         Then treat the left array partial sums.
      DO 25 KK=2,K
      DO 22 J=1,3
      XXL(KK,J)=0.D0
      XLL(KK,J)=0.D0
 22   CONTINUE
      KKM1=KK-1
      DO 25 I=1,KKM1
      IR = FF(I,KK)
      IF (IR .EQ. 0) GOTO 25
      CALL MMATML(K,R,I,IR,IR,B,XL,F)
      DO 23 J=1,3
      XXL(KK,J) = XXL(KK,J) + F(IR,J)
 23   XLL(KK,J) = XLL(KK,J) + XR(IR,J)
 25   CONTINUE
C
C     ... Next develop the right array partial sums.
C         Again the initial values must be zero.
      DO 30 KK=1,KM1
      DO 26 I=1,3
      XXR(KK,I)=0.D0
      XRR(KK,I)=0.D0
 26   CONTINUE
      KKP1=KK+1
      DO 30 J=KKP1,K
      IR=FF(KK,J)
      IF (IR .EQ. 0) GOTO 30
      CALL MMATML(K,R,J,IR,IR,B,XR,F)
      DO 27 I=1,3
      XXR(KK,I) = XXR(KK,I) + F(IR,I)
 27   XRR(KK,I) = XRR(KK,I) + XL(IR,I)
 30   CONTINUE
C
C     ...Collect the above quantities into the LHS of the system
C        of equations. Omit the end terms for now.
C
      DO 35 KK=2,KM1
      DO 31 II=1,3
 31   XL(K,II) = XLL(KK,II) + XRR(KK,II)
      CALL MMATML(K,R,KK,K,K,B,XL,F)
      DO 33 II=1,3
 33   LHS(KK,II)=XXL(KK,II)+XXR(KK,II)-F(K,II)
 35   CONTINUE
C
```

```
C       ...Finalize with the end correction terms.
C
        CALL MMATML(K,R,1,1,1,B,XRR,F)
        DO 38 I = 1,3
        LHS(1,I) = XXR(1,I) - F(1,I)
 38     CONTINUE
        CALL MMATML(K,R,K,K,1,B,XLL,F)
        DO 40 I=1,3
 40     LHS(K,I) = XXL(K,I) - F(1,I)
C
C       ... SOLVE THE LINEAR SYSTEM MA=L. This must be done three times
C           once for each column of LHS. Since the matrix M has rank K-
C           and since the solution is to be made relative to the first
C           array, we trim away the first row and first column of M to
C           get a non-singular system. For the same reason the elements
C           of the first row of A are all set to zero.
C
        DO 50 I=1,3
C       ... Prepare the input to SYSLIN for each column of LHS.
        DO 48 KI=2,K
        DO 45 KJ=2,K
        KIM1=KI-1
        KJM1=KJ-1
        AA(KIM1,KJM1)=M(KI,KJ)
 45     CONTINUE
 48     AA(KIM1,K)=LHS(KI,I)
C
C       ... Solve the system.
        CALL SYSLIN(AA,30,KM1)
        A(1,I) = 0.D0
C       ... Place the solution into the output matrix A.
        DO 50 KK=2,K
        KKM1=KK-1
        A(KK,I)=AA(KKM1,K)
 50     CONTINUE
C
C
        RETURN
        END


        SUBROUTINE MULMAT(BB,EE,EF)
C
C       ***************************************************************
C       ***  Subroutine to perform a linear transformation in  ***
C       ***  three-space.  Specifically, EF is the product of  ***
C       ***  the matrix BB and the vector EE.                  ***
C       ***************************************************************
C
        REAL*8 BB(3,3), EF(3), EE(3)
C
C       ... Initialize EE at zero:
        DO 10 I = 1,3
        EF(I) = 0.D0
 10     CONTINUE
```

```
      DO 20 I = 1,3
      X = 0.D0
C     ...  Accumulate the dot product of EE and the I-th row of BB:
      DO 30 J = 1,3
 30   X = X + BB(I,J) * EE(J)
      EF(I) = X
 20   CONTINUE
      RETURN
      END



      SUBROUTINE SYSLIN(A,IR,IC)
C
C     ***************************************************************
C     *** Finds the solution of a system of IC equations in IC    ***
C     *** unknowns.                                               ***
C     ***************************************************************

      REAL*8  A(1)
      ICP1 = IC + 1
      DO 107 K = 1,IC
      INDEX1 = (K-1)*IR + K
      A(INDEX1) = 1.D0/A(INDEX1)
      DO 102 J = 1,ICP1
      IF(J-K) 3,102,3
 3    INDEX2 = (J-1)*IR + K
      A(INDEX2) = A(INDEX2)*A(INDEX1)
 102  CONTINUE
      DO 115 I = 1,IC
      IF (I-K) 20,115,20
 20   INDEX3 = (K-1)*IR + I
      DO 114 J = 1,ICP1
      IF(J-K) 21,114,21
 21   INDEX2 = (J-1)*IR + I
      INDEX4 = (J - 1)*IR + K
      A(INDEX2) = A(INDEX2) - A(INDEX4)*A(INDEX3)
 114  CONTINUE
 115  CONTINUE
      DO 107 I = 1,IC
      IF(I-K) 22,107,22
 22   INDEX2 = (K-1)*IR + I
      A(INDEX2) = -1.D0 * A(INDEX2)*A(INDEX1)
 107  CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MMATML(K,R,L,M,N,B,E,F)
C
C     ***************************************************************
C     ***  Prepares for a linear transformation by a selected  ***
C     ***  Lth face of B; calls MULMAT to perform the          ***
C     ***  transformation on the Mth row of E and positions    ***
C     ***  the new vector in the Nth row of F.                 ***
C     ***  K = number of arrays in the problem                 ***
C     ***  R = number of rows in (common to) E and F           ***
C     ***  L = array index; first argument of B                ***
C     ***  M = index of the row of E                           ***
C     ***  N = index of the row of F                           ***
C     ***  B = Rank 3 family of rotational matrices            ***
C     ***  EE = multiplicand vector                            ***
C     ***  EF = product vector                                 ***
C     ***  E = input array (of rank 2), row M is used for EE   ***
C     ***  F = output array (of rank 2),EF is placed in row N  ***
C     ***************************************************************
C
C
      INTEGER*4 I, J, K, L, M, N ,R
      REAL*8 BB(3,3), B(30,3,3), EE(3), EF(3), E(30,3), F(30,3)
C
C     ... Initialize the temporary variables:
      DO 10 I = 1,3
      EE(I) = E(M,I)
      DO 10 J = 1,3
      BB(I,J) = B(L,I,J)
 10   CONTINUE
C
C     ... Call routine to perform linear transformation
      CALL MULMAT(BB,EE,EF)
C     ... Position the output as prescribed.
C
      DO 20 I = 1,3
      F(N,I) = EF(I)
 20   CONTINUE
C
      RETURN
      END


      SUBROUTINE DISPLC(K,B,A,IA,DEL,D)
C
C     ***************************************************************
C     *** DEL is the set of (output) displacements of the K arrays
C     *** estimated by the least square method.
C     *** D is the length of each row of DEL.
C     *** B is the set of K reorientation matrices (output of MULTAR).
C     *** A is the output of INTCEP (i.e. the array intercept vectors).
C     *** IA is the list of sensor arrays in the problem.
C     ***************************************************************
C
```

```
C     ... Declarations.
C
      REAL*8 B(30,3,3), A(30,3), AL(30,3), DEL(30,3), AA(3)
      REAL*8 ID(3,3), BB(3,3), D(30), ARNUM1, ARNUM2, ARNUM3, T(3)
      INTEGER*4 IA(30), K, DATE
C
C     ... Create an identity matrix ID:
C
      DO 10 I=1,3
      DO 10 J=1,3
      ID(I,J) = 0.D0
      IF ( I .NE. J ) GOTO 9
      ID(I,J) = 1.D0
 9    CONTINUE
 10   CONTINUE
C
C     ... Read file AR1.DAT in the order specified by IA.
C         This is used to create AL, the matrix of assumed locations
C         of those sensor arrays that appear in the problem.
C         The identication vector IA produced in CONECT is needed to
C         extract the correct rows from AR1. The result is AL(K,3).
C
      OPEN(2,FILE='AR1.DAT',STATUS='OLD')
      J = 1
 17   CONTINUE
      READ(2,*,END=18)  IAR, DATE, ARNUM1, ARNUM2, ARNUM3
      GOTO 15
 18   CONTINUE
      WRITE(*,*) ' Designated array number not found. '
      WRITE(*,*) ' Operation aborting. '
      STOP
 15   CONTINUE
      IF (IAR .EQ. IA(J)) THEN
          AL(J,1) = ARNUM1
          AL(J,2) = ARNUM2
          AL(J,3) = ARNUM3
          J = J + 1
          REWIND (2)
      ENDIF
      IF(J .LE. K) GOTO 17
C
      CLOSE (UNIT=2)
C     ... Compute the displacements.
C     ... First reduce by one the diagonal elements of each face of B:
      DO 30 KK=1,K
      DO 20 I=1,3
      DO 20 J=1,3
      BB(I,J) = B(KK,I,J) - ID(I,J)
 20   CONTINUE
C
C
C     ... Complete the displacement computation:
      DO 22 I=1,3
 22   AA(I) = AL(KK,I)
      CALL MULMAT(BB,AA,T)
```

```
      DO 25 I=1,3
      DEL(KK,I)= A(KK,I)+T(I)
 25   CONTINUE
 30   CONTINUE
C
C      ... Compute the length of each row of DEL.
C
      DO 40 KK=1,K
      D(KK) = 0.D0
      DO 35 I=1,3
      D(KK) = D(KK) + (DEL(KK,I) * DEL(KK,I))
 35   CONTINUE
 40   D(KK) = DSQRT(D(KK))
      RETURN
      END


        SUBROUTINE MAXANG(BB,P)
        REAL*8 B(3,3), BB(3,3), X(3), Y(3), D, P
C
C       ****************************************************************
C       *** Takes the orthonormal matrix BB as input and constructs ***
C       *** the angle of maximal rotation that any vector can       ***
C       *** experience under the transformation BB.                 ***
C       *** Calls MULT3.                                            ***
C       ****************************************************************
C
C       ... Begin with the adjustment of the diagonal elements of B
C           in order to prepare the eigenvector problem.
C
        DO 5 I=1,3
        DO 5 J=1,3
        B(I,J) = BB(I,J)
        IF (I .EQ. J) B(I,I) = B(I,I) - 1.
 5      CONTINUE
C       ... Use the first two equations of the eigenvector system
C           (with X3 = 1) to solve for the eigenvector.
C
C       ... Compute the determinant of the coefficient matrix:
        D = B(1,1)*B(2,2) - B(1,2)*B(2,1)
C       ... Check for zero rotation and transfer to the end
C           if it occurs.
        IF (D .EQ. 0.0) THEN
           D = 1.D0
           GOTO 30
        END IF
C
C       ... Set third component equal to one and solve the linear
C           system for the other two.
        X(3) = 1.D0
        X(1) = (B(1,2)*B(2,3) - B(1,3)*B(2,2))/D
        X(2) = (B(1,3)*B(2,1) - B(1,1)*B(2,3))/D
C
C       ... Normalize the eigenvectors.
C
```

```
      D = DSQRT(1.D0 + X(1)*X(1) + X(2)*X(2))
      DO 10 I=1,3
 10   X(I) = X(I)/D
C
C     ... Construct a vector orthogonal to the eigenvector.
C         First choose the subscript of the smallest X.
C
      J=1
      IF (DABS(X(1)) .GT. DABS(X(2))) J=2
      IF (DABS(X(J)) .GT. DABS(X(3))) J=3
C
C     ... Use the Gram-Schmidt technique to convert the direction
C         of X(J) to a direction orthogonal to X(1), X(2), X(3).
      DO 15 I=1,3
      IF (I .EQ. J) THEN
              Y(J) =  X(J)*(1.- X(J)*X(J))
      ELSE
              Y(I) = (-1.)*X(I)*X(J)*X(J)
      END IF
 15   CONTINUE
C     ... Normalize the new vector.
      D = DSQRT( Y(1)*Y(1) + Y(2)*Y(2) + Y(3)*Y(3) )
      DO 20 I=1,3
 20   Y(I) = Y(I)/D
C
C     ... Transform Y by BB and compute the angle between Y and BB*Y:
C
      CALL MULMAT(BB,Y,X)
      D = 0.D0
      DO 25 I=1,3
      D = D + X(I)*Y(I)
 25   CONTINUE
 30   P = DACOS(D)
      RETURN
      END


       SUBROUTINE ANGLES(K,B,P)
C
C      ***************************************************************
C      *** Computes the three Euler angles for each of the   ***
C      *** K matrices in B.                                  ***
C      ***************************************************************
C
       REAL*8 B(30,3,3),P(30,3)
C
       DO 20 KK=1,K
       P(KK,2)=DASIN (B(KK,3,1))
       P(KK,1)=DASIN (B(KK,3,2)/DCOS (P(KK,2)))
       P(KK,3)=DASIN (B(KK,2,1)/DCOS (P(KK,2)))
 20    CONTINUE
       RETURN
       END
```

DISTRIBUTION LIST

	NO. OF COPIES
Naval Undersea Warfare Engineering Station Code 70 Keyport, WA 98345	3
Naval Undersea Warfare Engineering Station Code 50 Keyport, WA 98345	5
Center for Naval Analyses 2000 Beauregard Street Alexandria, VA 22311	1
Operations Research Center, Room E40-164 Massachusetts Institute of Technology Attn: R. C. Larson and J. F. Shapiro Cambridge, MA 02139	1
Research Administration (Code 012) Naval Postgraduate School Monterey, CA 93943-5100	1
Library (Code 0142) Naval Postgraduate School Monterey, CA 93943-5100	4
Library (Code 55) Naval Postgraduate School Monterey, CA 93943-5100	1
Professor James Esary (Code 55Ey) Naval Postgraduate School Monterey, CA 93943-5100	1
Professor Robert Read (Code 55Re) Naval Postgraduate School Monterey, CA 93943-5100	20
Professor O. B. Wilson (Code 61W1) Naval Postgraduate School Monterey, CA 93943-5100	1